WL-TR-91-1042

(2)

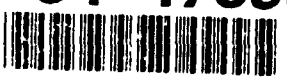
AGSSS: THE AIRBORNE GRAPHICS SOFTWARE SUPPORT SYSTEM; AN Ada/PHIGS-BASED DISPLAY EDITOR FOR THE RAPID DEVELOPMENT OF COCKPIT DISPLAY SOFTWARE SYSTEMS

R. Jorge Montoya, Timothy L. Turner, Donna M. Jewell,
James V. Aanstoos, Ramasubramanian Suresh, and M. Chad Barker
Center for Systems Engineering
Research Triangle Institute
Research Triangle Park, NC 27709

September 1991

Final Report for Period September 1987 - September 1990

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543



NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility nor any obligation whatsoever. The fact that the government may have formulated, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise in any manner construed, as licensing the holder or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Darryl A Williams

DERRYL A. WILLIAMS
Airborne Graphics S/W Support System
Program Manager
Systems Group

David A Zann

DAVID A. ZANN
Actg-Chief
Systems Integration Branch
Avionics Directorate

FOR THE COMMANDER

Charles H Krueger

CHARLES H. KRUEGER
Director
System Avionics Division
Avionics Directorate

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization, please notify WL/AAAS , Wright-Patterson AFB, OH 45433-6543 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS: None
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for Public Release: Distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): RTI/3966/00-01F	5. MONITORING ORGANIZATION REPORT NUMBER(S): WL-TR-91-1042
6a. NAME OF PERFORMING ORGANIZATION: Research Triangle Institute; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: WL/AAAS
6c. ADDRESS (City, State, and ZIP Code): Center for Systems Engineering, P.O. Box 12194, Research Triangle Park NC 27709-2194	7b. ADDRESS (City, State, and ZIP Code): Avionics Directorate, Wright Laboratory, Wright-Patterson AFB OH 45433
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable):	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F33615-87-C-1531
8c. ADDRESS (City, State, and ZIP Code):	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
63253F	2735	01	02

11. TITLE (Include Security Classification) AGSSS: The Airborne Graphics Software Support System; An Ada/Phigs-Based Display Editor For The Rapid Development Of Cockpit Display Software Systems

12. PERSONAL AUTHOR(S): R. J. Montoya, T.L. Turner, D. M. Jewell, J. V. Aanstoos, R. Suresh, M. C. Barker

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Sep 87 TO Sep 90	September 1991	9

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Software Development Tool, Airborne Graphics, Graphics Software Development, Dynamic Graphics

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Over the past decade, the performance of digital graphics systems has increased several-fold. At the same time, the size of the hardware has been reduced and high performance airborne systems are now feasible. As a result, complex, three-dimensional, pictoral, real-time display formats can now be supported. Unfortunately, the increase in display complexity results in a significant increase in the software requirements. Additionally, dynamic displays, such as used in avionics, have a two part software problem. First, the display format and all its elements must be explicitly defined. Second, the connection must be established to the rest of the avionics. Overall, graphics software development is a truly time and labor intensive task.

The goal of the AGSSS is to provide a graphics software development support environment. The AGSSS consists of four parts: the Graphics Editor for creating the format program, the Actions Editor for creating the driver software for the display dynamics, the Display Test Manager which allows the user to test the new software within the workstation, and the

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Derryl A. Williams	22b. TELEPHONE (Include Area Code): 513-255-4827	22c. OFFICE SYMBOL: WL/AAAS

DD Form 1473, JUN 86 — Previous editions are obsolete.

BLOCK 19 continued

Display Program Integrator which creates the final software source modules tuned to the target system hardware. The AGSSS is a totally interactive system which allows the user to create the display format by "drawing" the objects on a graphics workstation screen. The resulting code is shown, as it is produced, in an adjacent window. The user can operate in any window. The Actions Editor operates in a similar manner but employs text rather than graphics editing.

The system produces Ada/PHIGS code for the display file and Ada code for the actions file. Additionally, the AGSSS is written entirely in Ada providing a great deal of modularity and host system independence.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure No. **Page**

EXECUTIVE SUMMARY

This technical report addresses the development of the Airborne Graphics Software Support System (AGSSS), an integrated software development environment that aids in the rapid prototyping of cockpit displays and in the production of the associated display generation software. This work was sponsored by the Avionics Directorate of the Wright Laboratory (WL/AAAS) under contract F33615-87-C-1531. It was conducted by personnel of the Center for Systems Engineering (CSE), Research Triangle Institute (RTI). Mr. R. Jorge Montoya, manager of the Avionics Technology Department at RTI, served as the project manager and Mr. Derryl Williams of WL/AAAS-2 served as the project Engineer for the U. S. Air Force. He was ably assisted in this task by Mr. William Koenig. Mr. Jesse L. Blair, group leader, WL/AAAS-2 also helped set research goals and define AGSSS features. In addition to Mr. Montoya, the RTI AGSSS project team consisted of Mr. Timothy L. Turner, who led the technical effort, Mrs. Donna M. Jewell, Messrs. R. Suresh, James V. Aanstoos, and M. Chad Barker.

Advances in digital computers, computer graphics, and video technology during the last decade have made complex cockpit displays a common application in military aircraft. The underlying software system necessary to convert the data from the aircraft systems into a graphical representation, i.e., the display, is getting so complex that the application of traditional software methods to develop it is proving to be very inefficient. Classical coding methodology results in single point designs, promotes the development of the indispensable graphics programmer (Guru), and distances the display designer from the display implementation process.

To address these problems and encouraged by results from an earlier, proof-of-concept effort on the Interactive Graphics Editor (IGE)

performed under NASA contract NAS1-17948 and cosponsored by WL, the Avionics Directorate sponsored the research addressed in this technical report. The basic goals of the program defined in the PRDA 67-22-PMRB (CBD, 4/4/87) were: 1) to develop an interactive, pictorial tool which would bring the display designer closer to the display implementation process; 2) to make the application environment transparent to the display designer; and 3) to build in flexibility and avoid absolence in both the tool and its products.

In response to this PRDA, RTI proposed to develop a modular software environment that would be implemented in Ada and which would: 1) use the 3D PHIGS standard as the basis for the definition and rendering of the display formats, 2) use Ada as the basis for the specification of the actions that animate the resulting display, and 3) use Ada/PHIGS binding to implement the resulting real-time display software targeted to a specific application environment.

The result of the work performed under the ensuing contract (USAF F33615-87-C-1531) was the Computer Software Configuration Item (CSCI) known as AGSSS. AGSSS is an Ada/PHIGS-based, modular, software development environment implemented completely in Ada. It supports the development of display formats, the specification of the actions associated with the elements of such display formats, and the creation of a run-time software system to support the generation of the resultant cockpit display outside the development environment.

The implementation of the CSCI AGSSS consists of two main parts: 1) the design workstation and 2) the run-time system. The workstation consists of three Top Level Computer Software Components (TLCSCs), the AGSSS KERNEL, the DEVICE INTERFACE, and the DISPLAY EDITOR. The run-time environment is provided by the TLCSC RUN-TIME SUPPORT. A fifth TLCSC, AGSSS TOOLS, provides software components which are used by all other components of the system.

Specifically, the AGSSS KERNEL manages the workstation environment; the DEVICE INTERFACE interfaces AGSSS to all physical input and outputs.

The DISPLAY EDITOR supports the generation of display formats from pictorial definition through action specifications and testing to display program integration. The RUN-TIME SUPPORT provides building blocks for use with display programs generated by AGSSS and targeted to individual run-time environments. AGSSS TOOLS provides support packages of a general nature to all component levels of AGSSS.

Sample applications described in this report show that AGSSS has met or exceeded its design goals. Adherence to the 3D graphics standard PHIGS (Programmer's Hierarchical Interactive Graphics System) and implementation in the high-level language Ada insures portability and longevity of the product. Preliminary results indicate that use of the AGSSS has led to a significant shortening of the cockpit display design and development process in the Integrated Test Bed Facility at WL. Moreover, the AGSSS incremental Ada development environment shows the potential to improve substantially the efficiency of the display code development.

Informal measures indicate that by using the AGSSS the productivity associated with the process of developing cockpit displays and its enabling software improves tremendously, both in quantity and quality. The specialized knowledge required heretofore to work in this area has been reduced. Also, the system generates display programs which are targetable to other Ada/PHIGS environments. Examples of environments to which AGSSS outputs have been targeted include DEC VAXstation 3100 and SUN/4 SPARCStation 370.

Preliminary results indicate that productivity has increased by a factor of at least 10 over that obtained with the conventional method (FORTRAN/RAP) of developing display software for the ITB Facility Display Generation System (DGS). Furthermore, it is estimated that the integrated environment provided by AGSSS will improve productivity by a factor of 10 over hand-coding the same application in Ada/PHIGS and that further increases are possible with additional fine tuning of AGSSS. The Graphics Editor implementation is very thorough and has the potential for many more functions. It has certainly proved its worth as

a rapid prototyping tool supporting the iterative development process. The development of Ada programs has also benefited from the AGSSS implementation. For example, using the Actions Editor Ada program turn-around has improved between 10 and 50 times over the standard approach.

In addition to developing the AGSSS, efforts under this contract led to some promising, ancillary research results. These include the use of DIANA (the Descriptive Intermediate Attributed Notation for Ada), as the basis for an integrated Ada development system; the implementation of various extensions to the 3D graphics PHIGS standard; the development of a portable file manager; and the use of Ada tasking features to implement user inputs as concurrent finite automata.

The results of this project lead us to conclude that the potential is there for substantial quality improvements obtained by exploiting the iterative development process, the use of PHIGS, and the porting of AGSSS and its products to newer graphics platforms. Concerning the amount of specialized knowledge required to produce a cockpit display, significant reductions have been obtained by supporting the graphics specifications at a much higher level and implementing AGSSS to act as a guide with respect to the specifications of the actions in Ada. Finally, indications are that, with the exception of the generalized structure elements and the color specifications, the code generated by AGSSS will be 95% to 100% portable to other compatible platforms.

These results are sufficiently encouraging to lead us to recommend that AGSSS be ported to one or more of today's high-performance graphics workstations, especially the one chosen to upgrade the ITB Facility's DGS with. In addition the products of AGSSS should be targeted to the workstation of choice and a targeting strategy developed to exercise the application code produced by AGSSS in as many DGS as possible. Also subset configurations of AGSSS should be considered for other avionics applications. Specific enhancements to components of AGSSS should also be considered. These could include the addition of high-level primitives and PHIGS+ enhancements to the Graphics Editor; support for PDL and document generation in the Actions Editor; and provision of nonaerodynamic motion control in the Display Test Manager.

Longer term recommendations are based on exploiting the fact that the AGSSS has been designed and implemented with a great deal of modularity. For example, the KERNEL, the DEVICE INTERFACE, and the Ada TOOLS components of the AGSSS can be viewed as forming the basis for a generic Ada/PHIGS workstation which may be used to develop software for other, nongraphical, embedded applications. Furthermore, an implementation strategy should be followed that merges both the AGSSS windowing functionalities into those of X Windows, and the AGSSS PHIGS graphics with those of the emerging PEX (PHIGS Extensions to X) standard. This approach would promote the widest utilization of this productivity-enhancement tool.

1.0 INTRODUCTION

1.1 Purpose

This technical report describes the development of the Airborne Graphics Software Support System (AGSSS), an Ada/PHIGS-based software development environment for the rapid prototyping of cockpit displays and the automatic production of their enabling software. The work was sponsored by the Avionics Directorate of the Wright Laboratory (WL/AAAS-2) under contract F33615-87-C-1531 and conducted by personnel of the Center for Systems Engineering (CSE) of the Research Triangle Institute (RTI).

The purpose of AGSSS is to support advanced airborne display generation systems efficiently and with sufficient flexibility to accommodate future display systems requirements. It is intended to give the cockpit display designer full but transparent access to the display generation environment by providing interactive interfaces that will allow the artistic or pictorial design of displays, guide the specification of the operation or the dynamics of the display, and automatically create the run-time code that will generate the display formats in the airborne environment.

1.2 Need

Development of software for cockpit displays poses a tremendous challenge to display designers and software implementors. Recent technological advances have produced computers which are smaller, less power-consuming, and which have higher computational power and memory capacity than ever before. These advances have made their way into today's high-performance airborne display systems. These systems are so complex and their application-support software so intricate that they require experts to program them. Moreover, in most of these cases, software development tools have not kept pace with this rapid hardware evolution.

In addition, the increasing acceptance of color, raster scan display technology in airborne applications has created both benefits and problems as display designers seek to exploit its many capabilities for encoding and integrating information and display implementors seek ways to program the display systems efficiently to generate these displays in real time.

Cockpit display formats have evolved from 2D and 3D replicas of standard cockpit instrumentation to very sophisticated 3D images which integrate a great deal of information and detail into one display. Figures 1.1 through 1.4 illustrate the level of complexity found in today's representative cockpit display. Figure 1.1 shows a replica of a horizontal situation indicator (HSI) in the TACAN mode. Figure 1.2 illustrates a vertical situation display (VSD). This display is slightly more complicated than the TACAN display. Figure 1.3 includes several of today's representative cockpit display formats. Of particular importance to this discussion are the two formats at the bottom of the figure. The display on the left illustrates a primary flight display (PFD) known as the electronic attitude director indicator (EADI) which has been integrated with a tunnel-in-the-sky display. The display on the right illustrates the EADI integrated with a synthetic 3D scene. Figure 1.4 illustrates the Pathway-in-the-Sky (PITS) display, a follow-me display concept currently under consideration for addition in the ITBF research cockpit. The displays in the last two figures rank among the most complex found in today's cockpit. A number of conceptual cockpit display formats for future applications have been discussed in the literature. As a rule they seem to be between one and two orders of magnitude more complex than the ones presented in Figures 1.3 and 1.4. One example of this is the format associated with the "super cockpit display" identified in Project Forecast II. This futuristic cockpit display integrates a great deal of information into one format and should be representative of displays of the future.

The typical 3D display format consists of a collection of 3D geometric primitives such as polygons, lines, and points.

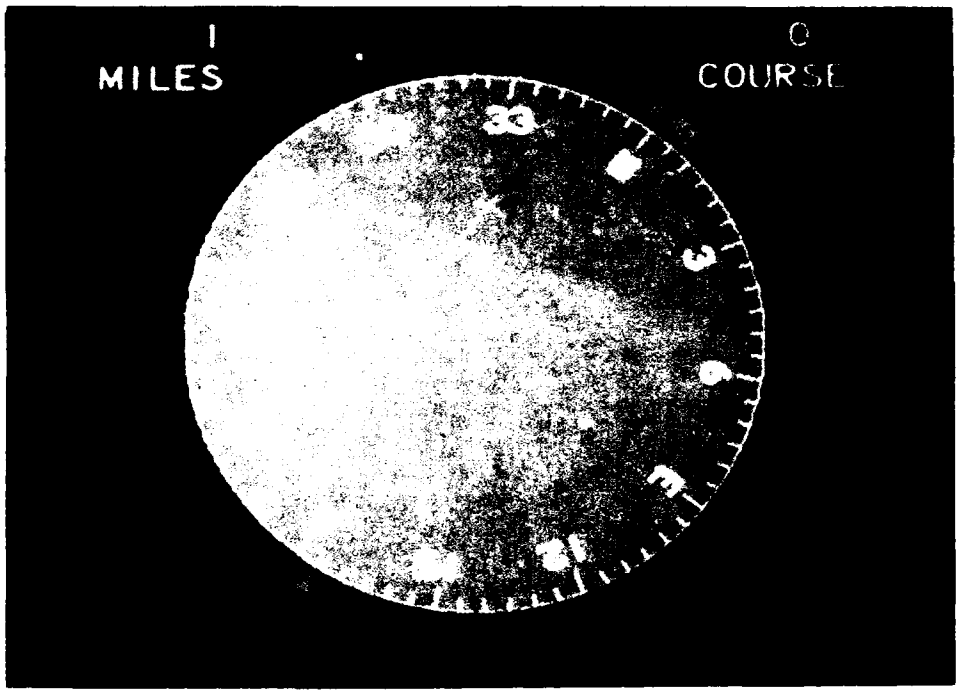


Figure 1.1 Horizontal Situation Indicator: TACAN Format

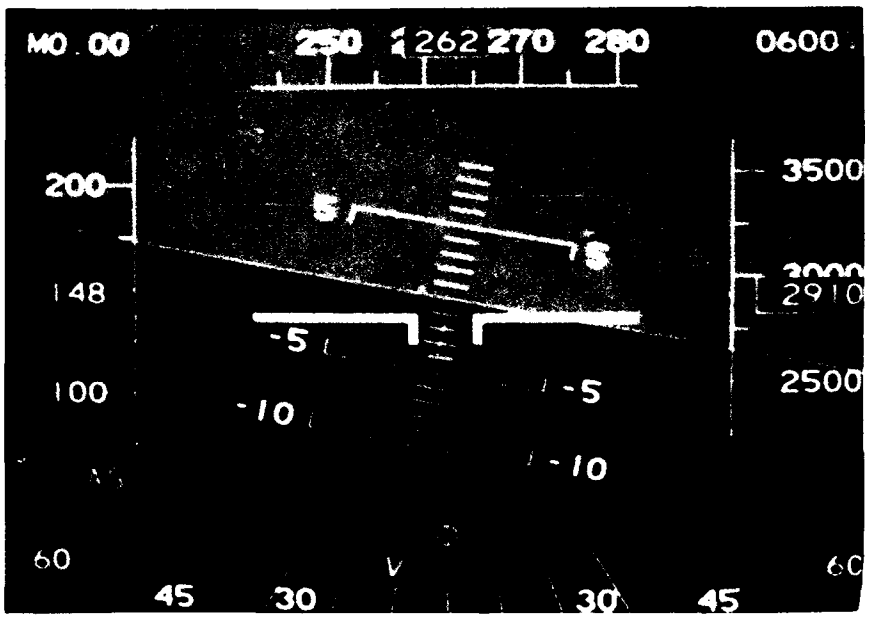
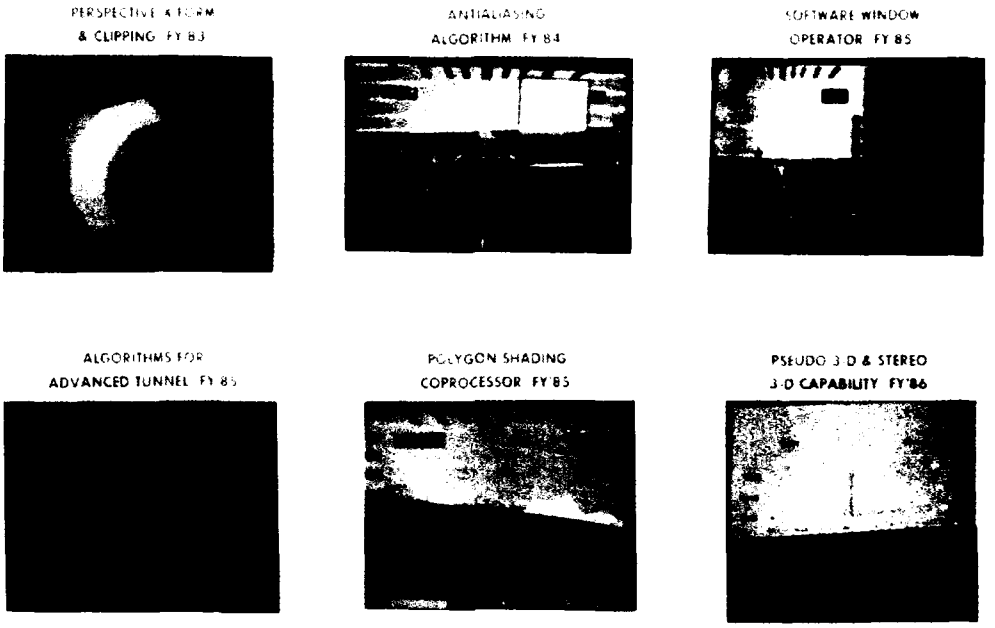


Figure 1.2 Vertical Situation Display (VSD)

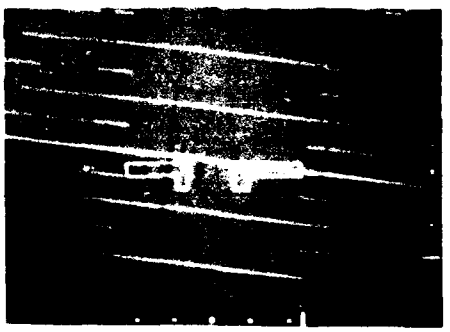
SOFTWARE/HARDWARE BASE FOR ADVANCED FLIGHT DISPLAYS



PSEUDO 3-D SCENE GENERATION

FEATURES MERGED WITH EADI

"TUNNEL IN SKY"
PLUS RUNWAY



ADAGE 3000 WITH
HARDWARE FILL

- 512 X 512 RESOLUTION
- 25 MILLION PIXELS/SECOND

"REAL-WORLD"TERRAIN
DATA BASE

ADAGE 3000 WITH
APEX CO PROCESSOR

- 1024 X 1024 RESOLUTION
- 40 MILLION PIXELS/SECOND

Figure 1.3 Cockpit Displays With Varying Degrees of Complexity

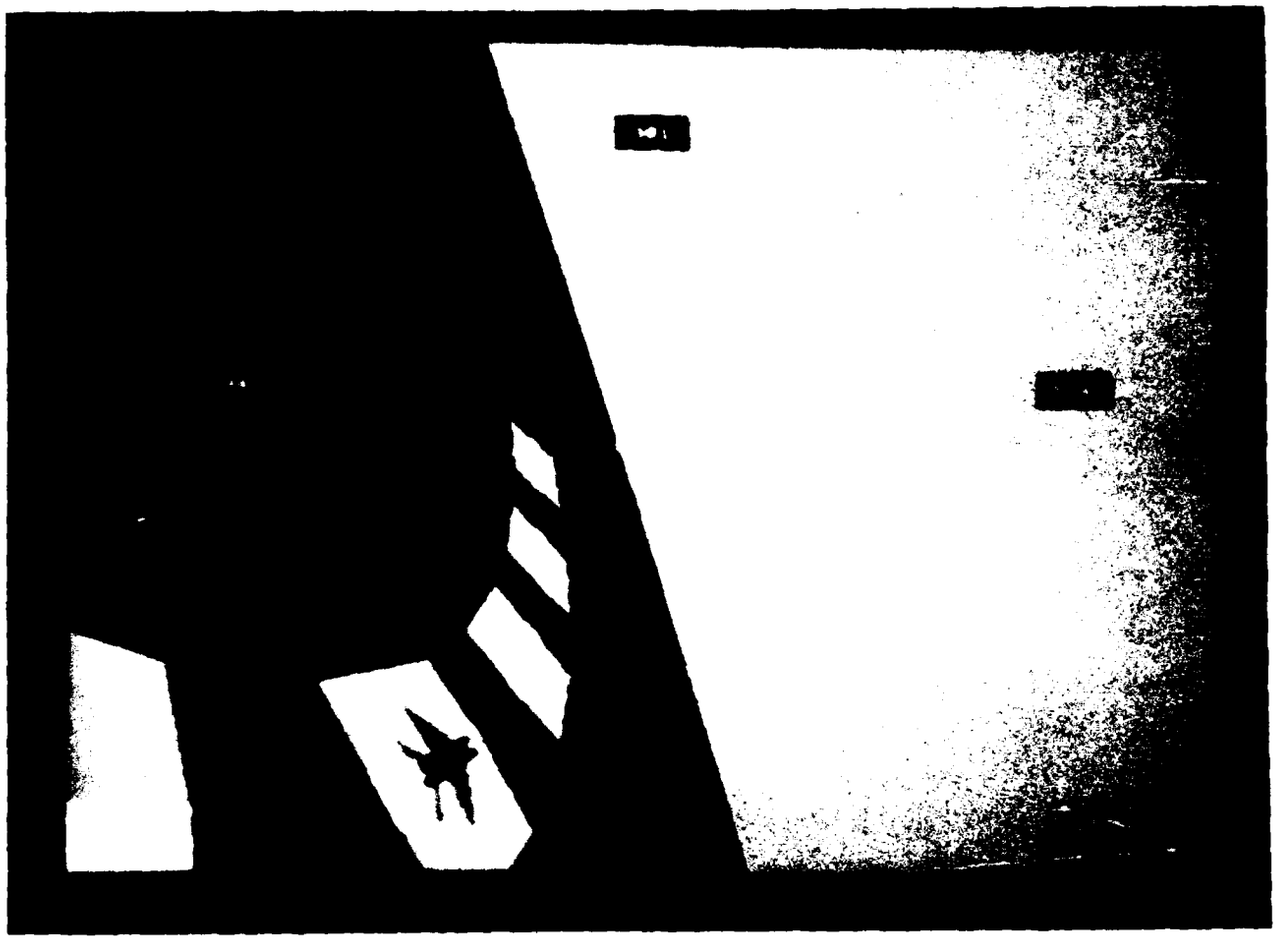


Figure 1.4 Pathway-in-the-sky (PITS) Display Format

Fundamentally, any display format can be decomposed into these geometric primitives. The combination of these primitives with the update rate required by the application provides a metric with which the display designer can predict, at least in general terms, the performance characteristic required of the associated display system. Although alpha-numeric displays are rather innocuous, text and its associated attributes (e.g., antialiased and rotatable) add substantially to the computational load presented by a display format to the display generator, specially in display systems which are not optimized for character generation. Furthermore, these displays will have smooth shading requirements which will introduce an additional level of complexity in the generation of the formats.

Furthermore, these displays will incorporate a great deal of modality which will increase the complexity of the software. Additional display software complexity will result from the management of critical cockpit displays used in a particular aircraft. This will have to be done increasingly in the future to fully utilize the limited amount of display real estate which will be available in the cockpit.

In the context of cockpit displays, real-time performance is usually taken to mean display update rates of 30 Hz or better. This rate indicates how often an image of a given complexity must be updated to give the pilot the assurance that the information is indeed coming from the real world. It should be clear that for a given graphics computer, the higher the complexity of the display and its associated software, the slower (longer) the update rate (the execution time) of the display (the program) will be. Partly because of this, software developers are constantly searching for efficient ways of accomplishing a given graphical procedure. This consideration also provides the impetus for hardware designers to incorporate hardware-assist capabilities in graphics computers. Sometimes it is possible to implement graphical procedures more efficiently in hardware resulting in significant performance increases. For example, the ADAGE 3000, the PDG in the ITB Facility Display Generation System (DGS) provides several coprocessors and special hardware features. The hardware fill feature of this PDG,

for example, allows for the efficient flat shading of polygons. Coprocessors are also available to perform a variety of functions including text generation and rendering which is a particularly inefficient task in raster scan PDGs.

Because the enabling software for these display systems is so focused and their hardware environment is so unique, the application of conventional software development techniques in this environment has not been very productive. This problem is compounded by the inability of the display designer, typically not an experienced programmer, to exploit directly the power of these computers. These factors result in long display development periods during which the designer has no feedback as to what his or her concept will look like in the application. Consequently, the development of cockpit displays using the classical approach has become very costly and time consuming.

Encouraged by results from earlier research on the Interactive Graphics Editor (IGE) (R 1), the Avionics Directorate of the Wright Laboratory (WL/AAAS-2) of the U.S. Air Force sponsored the Research Triangle Institute (RTI) to conduct research into the definition and implementation of a software system that would make the development of cockpit displays and its underlying software a more efficient process.

As described in the Program Research and Development Announcement (PRDA) 67-22-PMRB which appeared in the Commerce Business Daily (CBD) dated April 4, 1987, "the objective of the Airborne Graphics Software Support System (AGSSS) is to support advanced airborne display generation systems and will be designed with sufficient flexibility to accommodate future display systems requirements without becoming obsolete. The system will be hardware independent, generating generic and standard software code which can be targeted to many differing display systems. The system will support graphics software development and modification interactively and will automate many of the procedural aspects of the development process, allowing the designer to concentrate his/her efforts on the creative aspects of the process." A block

diagram of the AGSSS concept as envisioned at the time the PRDA was announced is presented in Figure 1.5.

1.3 Approach

In response to this PRDA announcement, the Research Triangle Institute proposed to the Avionics directorate of WL (WL/AAAS-2) a research and development program to develop a PHIGS/Ada-based software system which would provide a pictorial ("artistic" versus "codable") interface to display designers in the front end and produce generic graphics and application code in the back end. The resulting display code could be targeted to existing and projected suite of airborne processors (display and host) to support the generation and update of sophisticated cockpit displays. As part of the program, RTI also proposed to demonstrate the software system in the hardware and software environment of the WL/AAAS's ITB facility. This facility provides an integrated environment in which to evaluate and demonstrate emerging avionics hardware and software systems. A block diagram of the ITB Facility (ITBF) is illustrated in Figure 1.6 and a picture of the research cockpit in the ITBF is included in Figure 1.7. It was also specified that the resultant software system would be used in the ITBF as a rapid display prototyping tool and as a software generator tool after its demonstration phase. As such, the product would be designed and implemented to adhere to all specified data items (DI) standards referred to in the PRDA announcement. A block diagram of the AGSSS concept proposed by RTI is presented in Figure 1.8.

Based on this response, a contract was awarded to RTI in September 1987 to refine the definition of the system and conduct the necessary research to develop the Airborne Graphics Software Support System (AGSSS). The objective of the work described in this report was to develop a modular software development environment for designing, modifying, and testing cockpit displays, and automatically creating the underlying programs in a high-level and interactive manner. A fundamental goal of the software system was to address the problem of

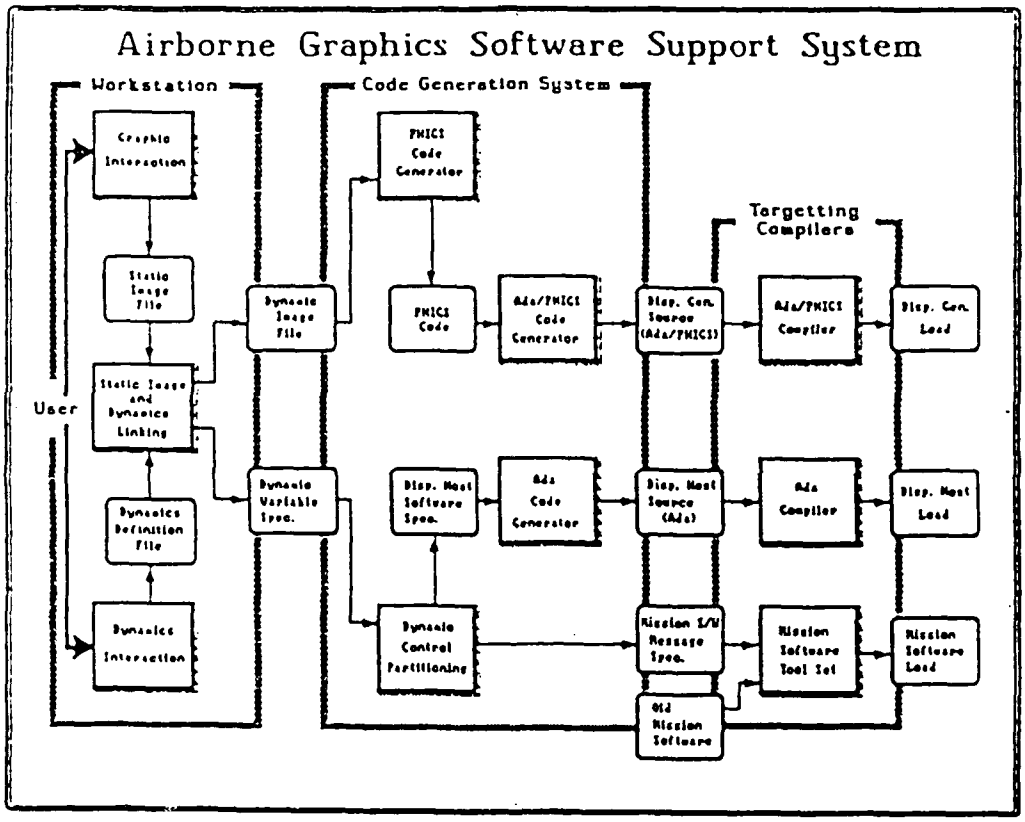


Figure 1.5 Block Diagram of Original AGSSS Concept

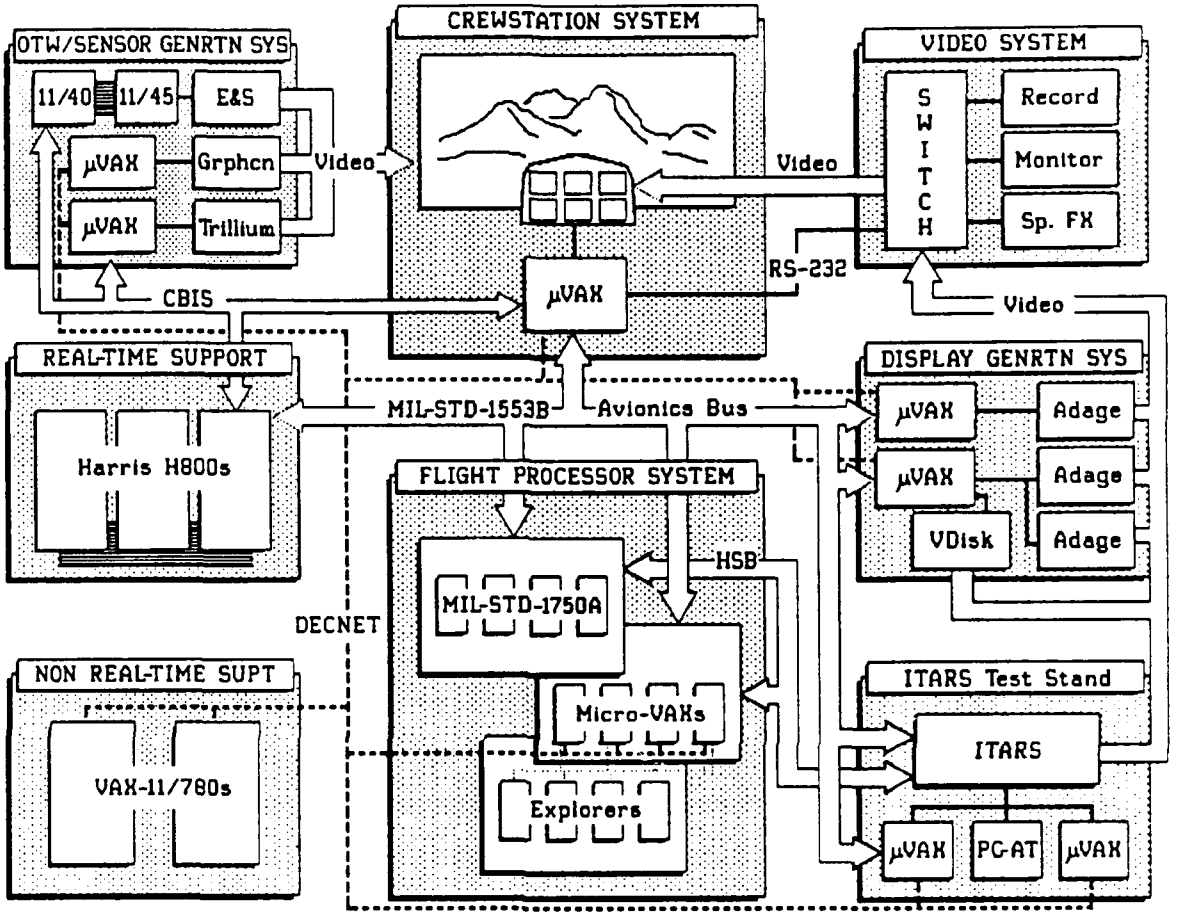


Figure 1.6 WL Integrated Test Bed (ITB) Facility

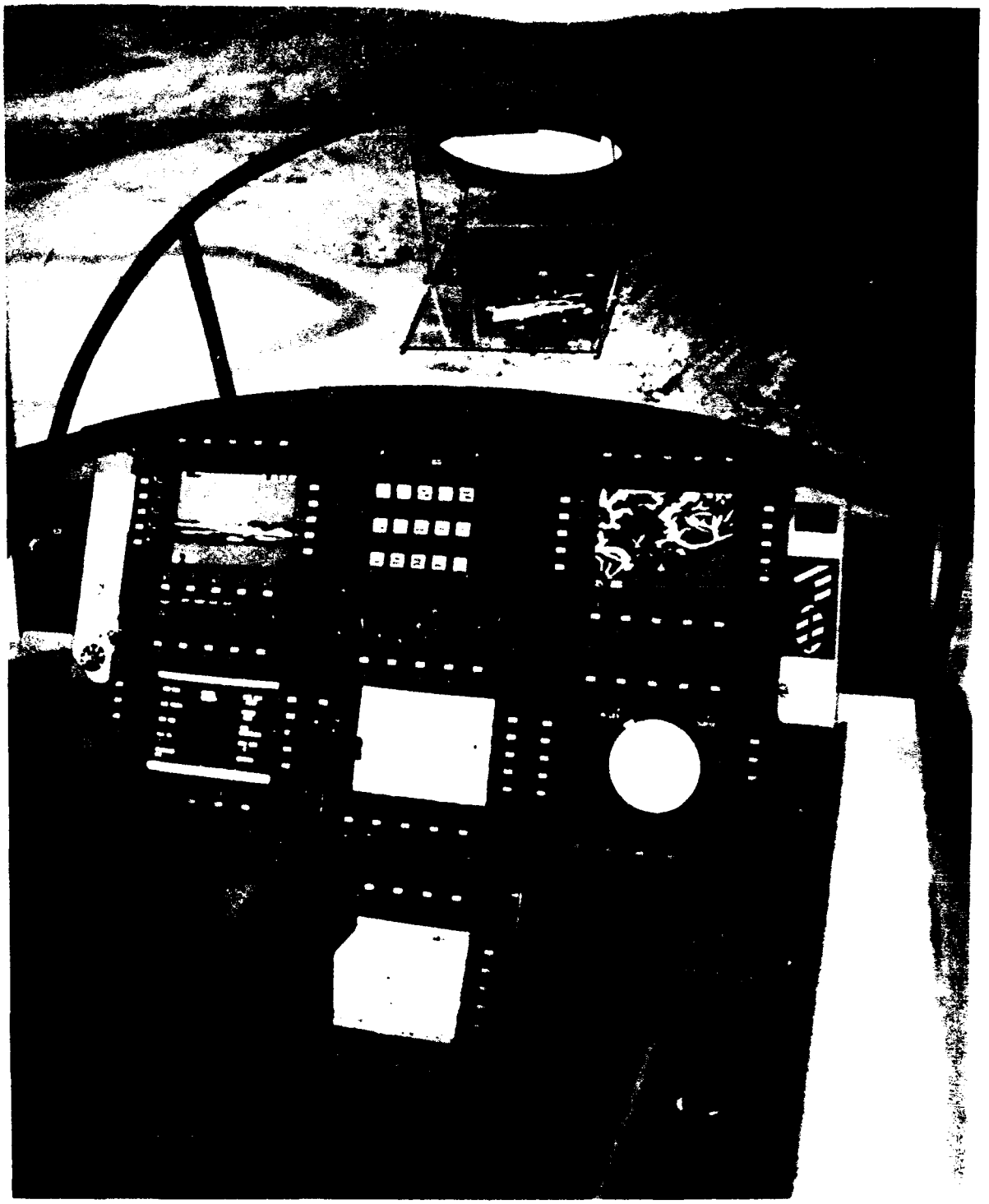


Figure 1.2 Research Cockpit of the ITB Facility

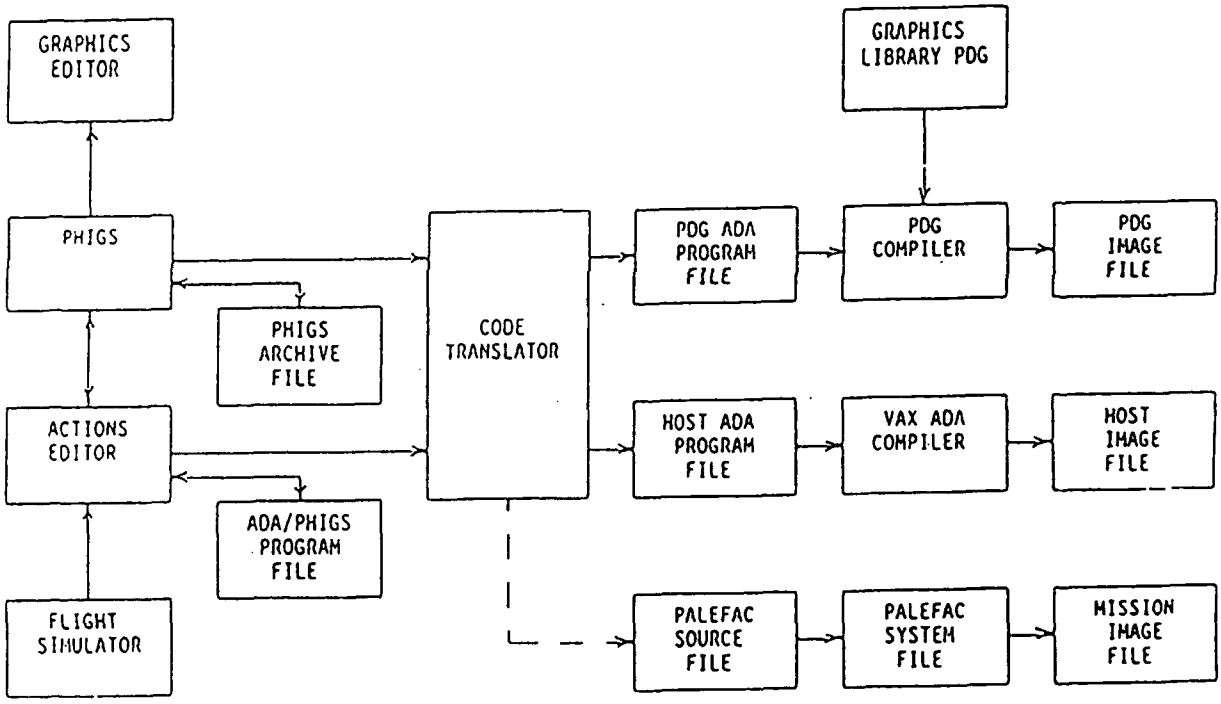


Figure 1.8 Block Diagram of Proposed AGSSS Concept

developing software for graphics generation across a varied population of airborne display systems, both current and projected. This required an implementation strategy based, to the fullest extent possible, on standard programming languages and graphics procedures. Functionally, the AGSSS supports the interactive and artistic creation of cockpit display formats through a PHIGS-based graphics editor, the interactive specification of the dynamics of the display symbology through an Ada-based actions editor, and the incremental testing of the emerging display through a display test manager. AGSSS uses the resultant object definitions (in PHIGS) and display actions specifications (in Ada) to automatically generate properly partitioned display system code (in Ada and Ada/PHIGS) for a target display system.

PHIGS, the Programmer's Hierarchical Interactive Graphics System, is a 3D graphics standard (R 2) which defines a graphics support system to control the definition, modification, storage, and display of hierarchical graphics data. Associated with PHIGS are language bindings, which are specifications of the interface between an application program and the PHIGS implementation in a given high-level language. Since the AGSSS is implemented in Ada (R 3) and all high-level code produced by the AGSSS is in Ada, it was necessary to implement the PHIGS binding to Ada (R 4) in the display system.

This report describes the implementation of the AGSSS, its integrated operation, and its use. The report also includes the description of some significant innovations emanating from the project including the development of Ada language tools and the implementation of concurrent PHIGS. The report also contains a series of conclusions stemming from the use of AGSSS to date and recommendations about possible extensions to and additional applications of AGSSS. A Conceptual block diagram of the implementation of the AGSSS is presented in Figure 1.9.

Major milestones of this project include Preliminary Design Review (PDR) on July 7-8, 1988, Critical Design Review (CDR) on November 15-17, 1988, AGSSS minimum version delivery and demonstration on October 16-17, 1989, AGSSS final version delivery and demonstration on July 23, 1990,

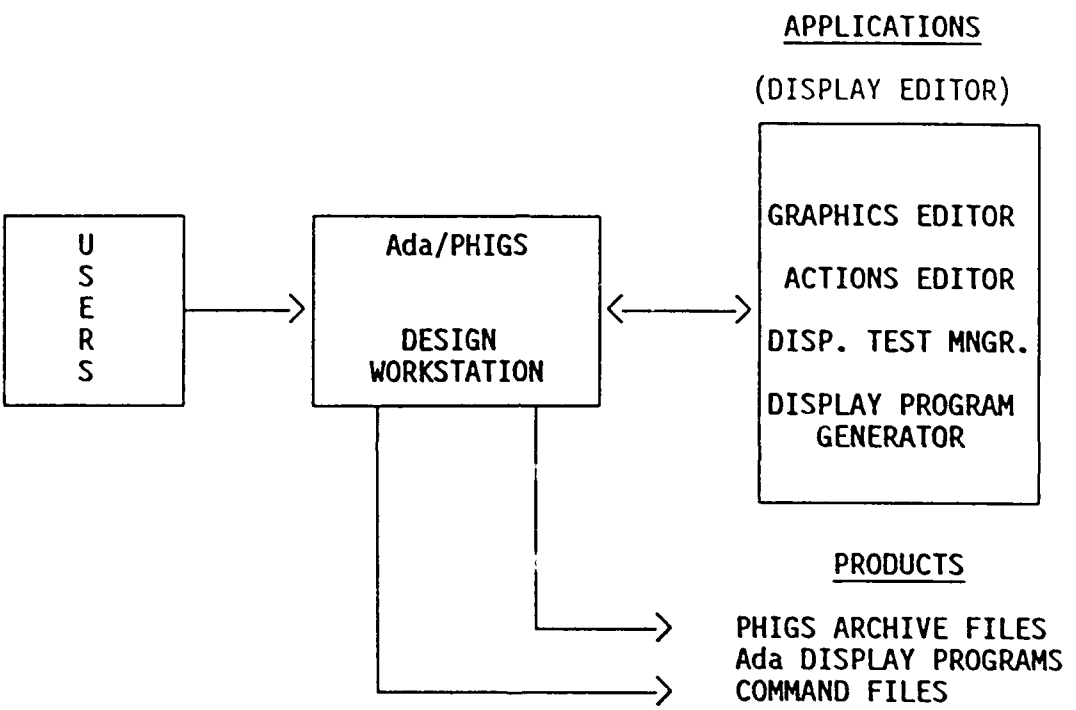


Figure 1.9 Functional Block Diagram of AGSSS

and AGSSS training session on September 24-28, 1990. Documents generated and delivered under this contract include Software Top Level Design Document for the AGSSS (CDRL #6), Software User's Manual for the AGSSS (CDRL #7), Software Development Plan for the AGSSS (CDRL #8), and Software Detailed Design Document for the AGSSS (CDRL #9). Two other documents not identified in the CDRL list were generated and delivered in draft form. They are: Systems Analysis Document for the AGSSS and Software Configuration Control and Reporting System (SCCRS) User's Manual. In addition, a paper entitled "An Ada-Based, Portable Design Workstation for Computer-Generated Cockpit Displays" was presented at the 9th Digital Avionics System Conference (R 5).

2.0 AGSSS DESIGN, IMPLEMENTATION, AND USAGE

This section describes the detailed design of the Computer Software Component Item (CSCI) identified as the Airborne Graphics Software Support System (AGSSS) of the AGSSS software system. As previously stated, the purpose of the AGSSS software system is to provide a PHIGS/Ada-based software development system which will support the interactive and pictorial development of cockpit displays and their dynamic specifications; create generic and machine-independent, high-level language display programs, and compile these programs into target airborne programmable display generators (PDGs) and host processors. The specific software modules have been developed in adherence with the PHIGS graphics standard, the Ada DoD/ANSI standard, and the PHIGS/Ada binding.

The AGSSS has been developed for and demonstrated in the display system of the WL's Integrated Test Bed (ITB) Facility. The AGSSS will be used as a rapid display prototyping tool and as a display system software generator tool in this facility. The AGSSS hardware environment in the ITB Facility consists of a front-end PC/AT workstation (Zenith 248), a MicroVAX III host computer, and an Adage 3000 Programmable Display Generator (PDG) (R 6). The corresponding software environment consists of the MS-DOS vers. 3.3 operating system in the PC, VMS vers. 5.3 operating system in the MicroVAX, and Adage microcode in the PDG. Specific Ada compilers in both the PC and the MicroVAX support the AGSSS implementation.

As part of this project, several workstations have been identified as possible platforms to port AGSSS and/or to target its outputs to. Because these workstations implement the PHIGS rendering model to various degrees of completeness, the need for a machine-specific, back-end cross-compiler most likely will disappear in the future. Instead, a strategy for the targeting of AGSSS graphics outputs will be developed that will seek to interface to the particular PHIGS implementation through the specific Ada language binding of each platform.

2.1 Detailed Design

The AGSSS design goals were set to: 1) obtain system flexibility to accommodate future changes, 2) support interactivity in the development of displays and the specification of their dynamics, and 3) support the automatic generation of display programs and their targeting to different display systems.

AGSSS supports the pictorial definition of display formats through a PHIGS-based Graphics Editor, the interactive specification of display format dynamics through an Ada-based Actions Editor, the incremental testing of the evolving display format with the aid of the Display Test Manager, and the generation of display code in Ada and Ada/PHIGS through the Display Program Generator. Use of Ada and adherence to PHIGS allow AGSSS to import foreign pictorial data and Ada programs. These factors also promote the portability of the tool and the targetability of its products.

The AGSSS system consists of two main parts,

1) The AGSSS design workstation, and
2) The run-time systems.

The first part provides the system which the display designer uses to develop the displays. The second part provides the run-time environment which the displays will run in.

The AGSSS design workstation uses:

The AGSSS KERNEL,
The DEVICE INTERFACE, and
The DISPLAY EDITOR.

The run-time environment is provided by

RUN-TIME SUPPORT.

A fifth component,

AGSSS TOOLS,

provides software components which are used by all the other components of the system.

Figure 2.1 shows an input-output description of the AGSSS design workstation. It generates a collection of files for use by the run-time systems. Figure 2.2 shows the workstation decomposed into its component Top Level Computer Software Components (TLCSCs). Some of the files generated by the design workstation are used to generate the executable image for the run-time system. Two run-time modes are supported:

1) A stand-alone simulator, called the Display Demonstrator, and

2) A display generation system which communicates with the airborne computer system over an avionics bus.

Figure 2.3 shows how the Display Demonstrator is created. Figure 2.4 shows how the Mission Display Generation System is created.

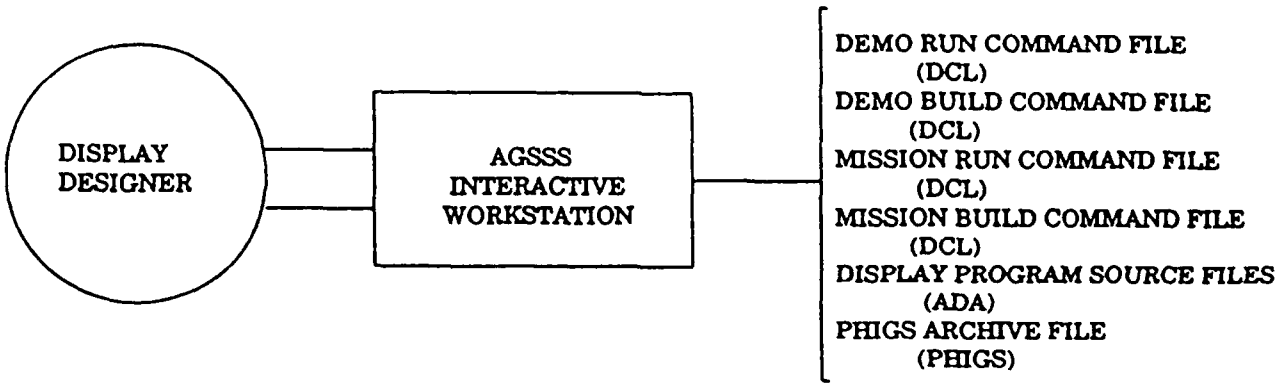
In the AGSSS workstation, three control modes are identified:

1) Initialization/Shutdown,
2) Workspace control, and
3) Functional control flow.

The three types of control flow are illustrated in Figures 2.5, 2.6, and 2.7, respectively.

2.2 Implementation

The AGSSS implementation meets the goal of providing a tool for designing, modifying, and testing graphics display programs in a high-level and interactive manner. The user communicates with the AGSSS through a powerful and flexible graphical user interface (GUI) which supports a variety of input and output devices. Layered on top of this interface are the key "applications" which are highly integrated to meet



DCL DIGITAL COMMAND LANGUAGE (VMS)
PHIGS PROGRAMMER'S HIERARCHICAL INTERACTIVE GRAPHICS SYSTEM

Figure 2.1 CSCI 1 AGSSS System Diagram

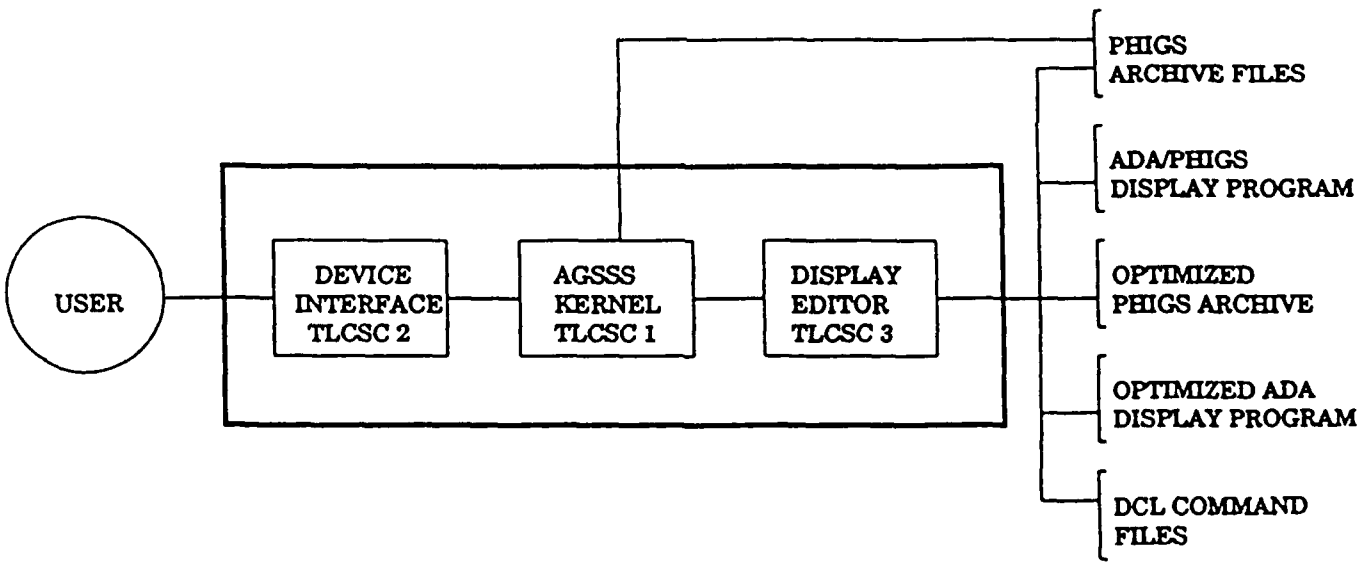


Figure 2.2 CSCI 1 File Level Data Flow

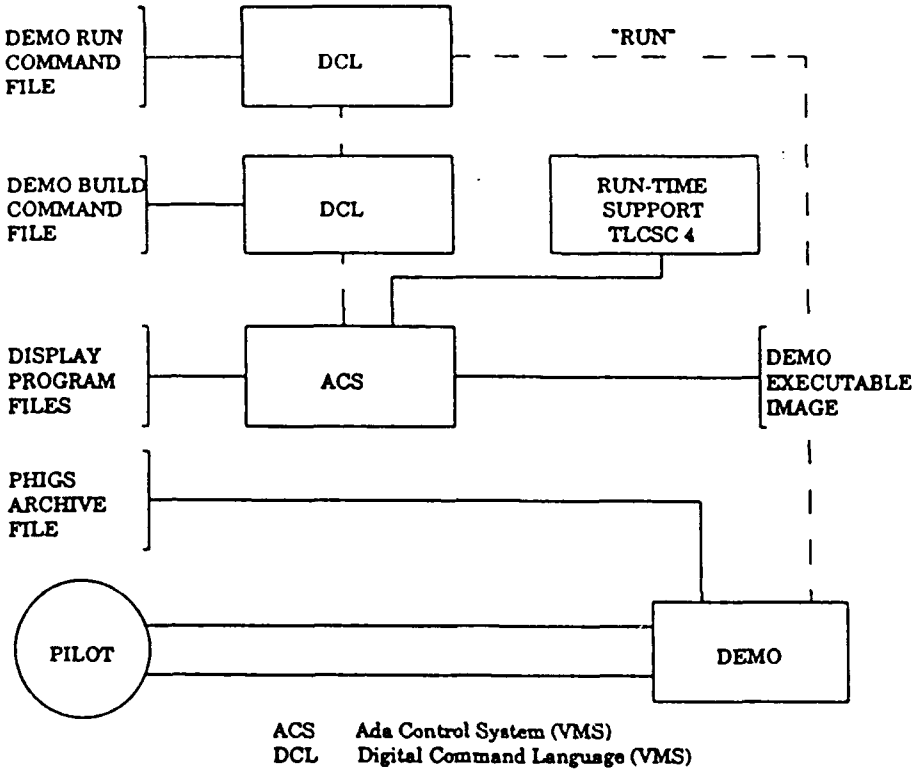


Figure 2.3 CSCI 1 AGSSS Demonstrator Mode

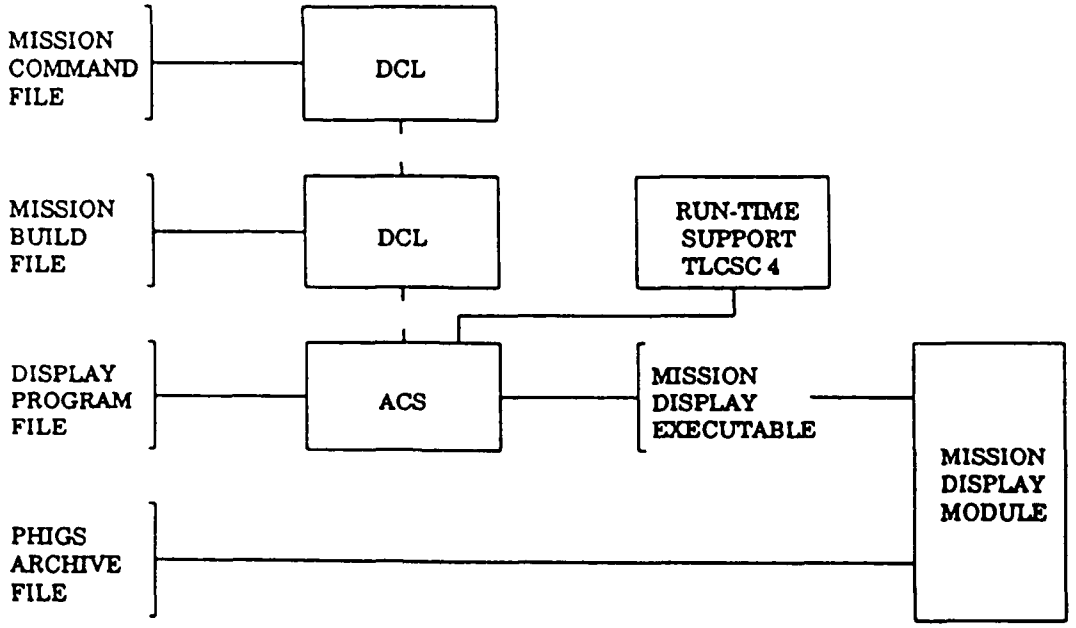


Figure 2.4 CSCI 1 AGSSS Mission Mode

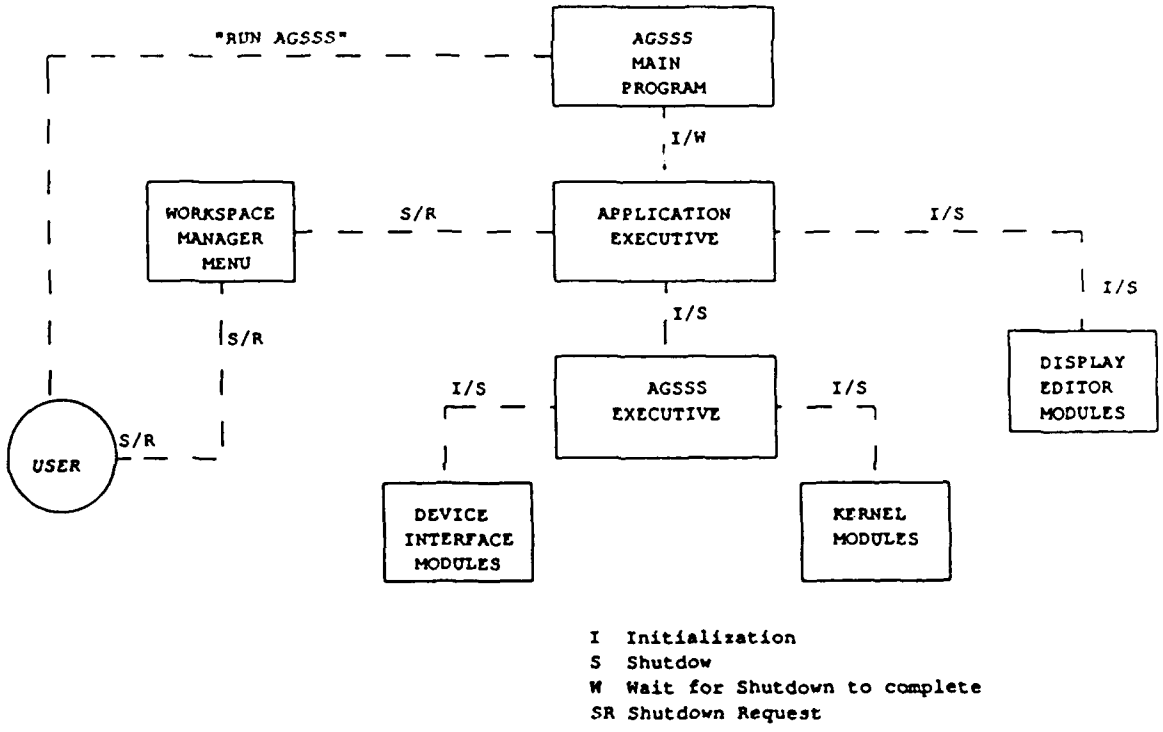


Figure 2.5 CSCI 1 AGSSS Control Flow:

Initialization/Shutdown

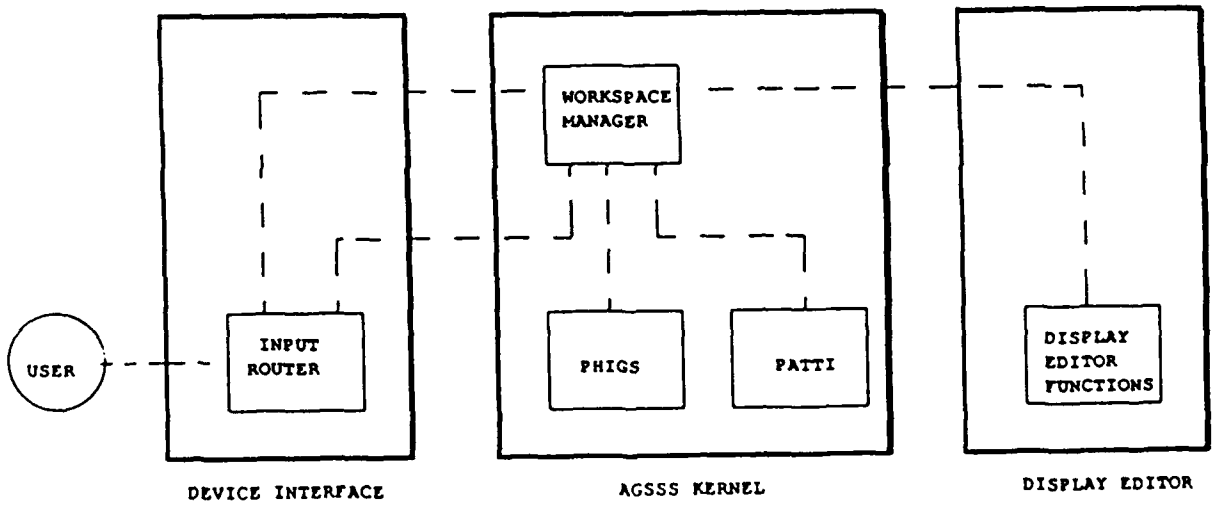


Figure 2.6 CSCI 1 AGSSS Workspace Control Flow

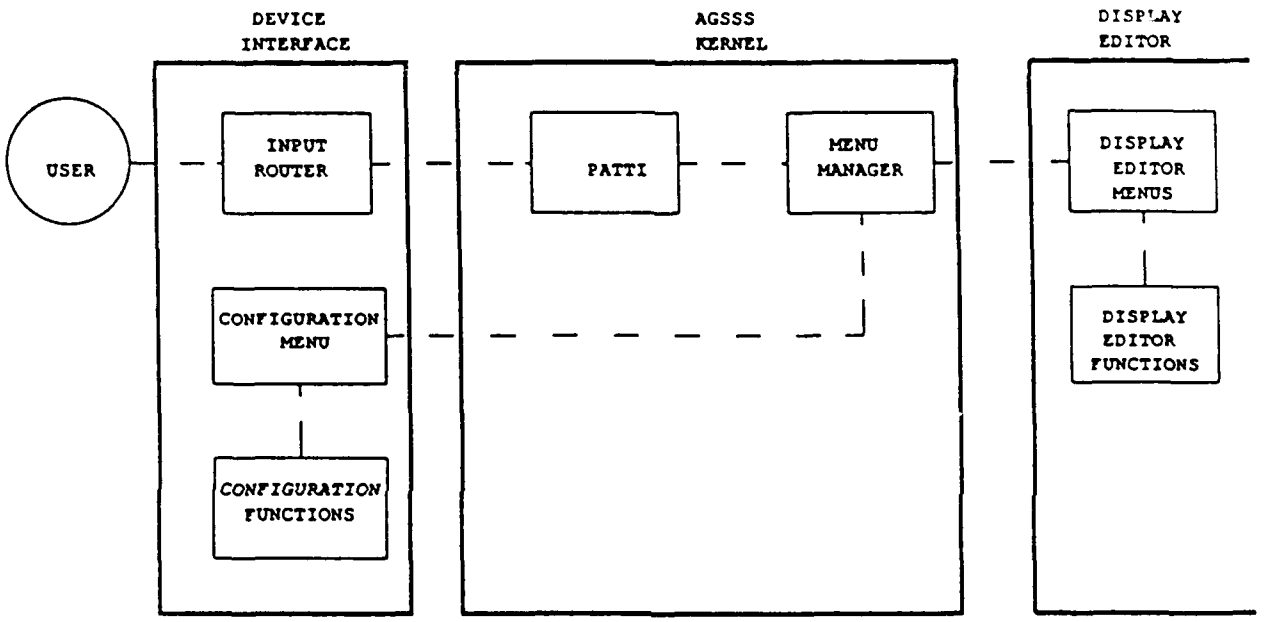


Figure 2.7 CSCI 1 AGSSS Functional Control Flow

this goal: The Graphics Editor, the Actions Editor, the Display Test Manager, and the Display Program Generator. Integrated within the system are four additional utility applications which support the roles of the key applications: The Color Editor, the File Manager, the Text Editor, and the Workspace Manager. At the heart of the system is an RTI implementation of the 3D graphics standard PHIGS, the Programmer's Hierarchical Interactive Graphics System, which has been extended to support concurrent calls from multiple tasks. The system is comprised of the Device Interface, the AGSSS Kernel, the Editors, and the Ada tools. The AGSSS Kernel provides generic Ada/PHIGS services to the applications. Ada tools are used throughout the implementation. These modules and their interrelations are illustrated in Figure 2.8. User interactions take place from the left and systems services (applications) take place from the right. This figure shows a functional block diagram of the AGSSS as currently implemented.

The AGSSS design has been implemented in five Top-Level Computer Software Components (TLCSCs). They are identified as follows:

1) AGSSS KERNEL (TLCSC 1),
2) DEVICE INTERFACE (TLCSC 2),
3) DISPLAY EDITOR (TLCSC 3),
4) RUN-TIME SUPPORT (TLCSC 4), and
5) AGSSS TOOLS (TLCSC 5).

The top-level composition of the AGSSS CSCI is presented in Figure 2.9.

The function of the AGSSS KERNEL (TLCSC 1) is to manage the AGSSS workstation environment. The AGSSS KERNEL consists of 11 Low-Level Computer Software Components (LLCSCs):

1) Kernel Executive (LLCSC 1.1),
2) Workspace Manager (LLCSC 1.2),
3) PHIGS Module (LLCSC 1.3),
4) PATTI (Programmers Attributed Text Interface) (LLCSC 1.4),
5) Menu Manager (LLCSC 1.5),
6) Forms Manager (LLCSC 1.6),
7) Color Editor (LLCSC 1.7),
8) File Manager (LLCSC 1.8),
9) Message Logger (LLCSC 1.9),
10) Dialog (LLCSC 1.10), and
11) Text Editor (LLCSC 1.11).

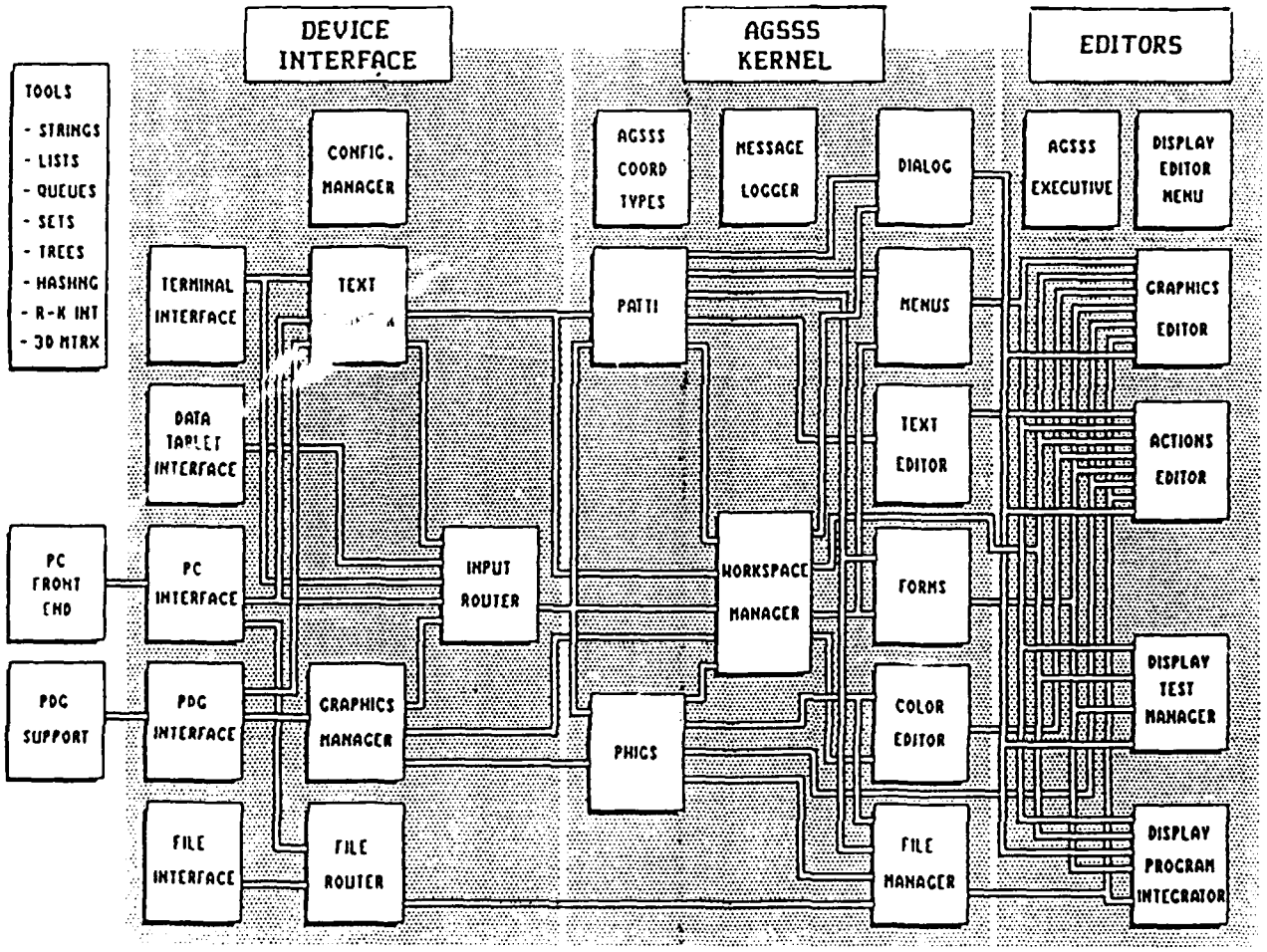


Figure 2.8 Block Diagram of AGSSS Modular Implementation

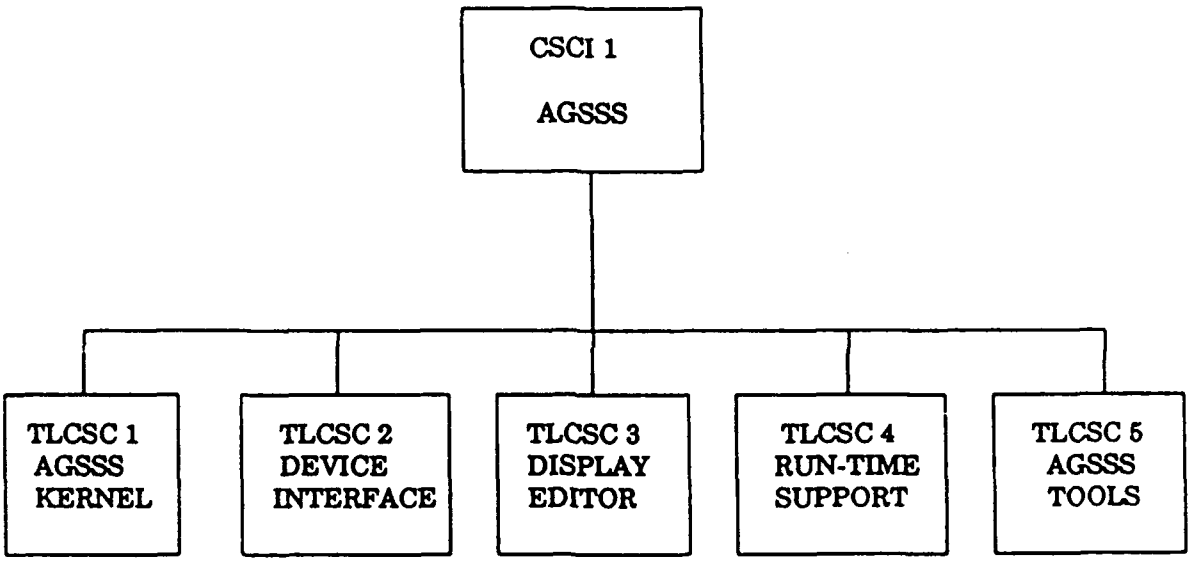


Figure 2.9 CSCI 1 Composition of the AGSSS

The function of the DEVICE INTERFACE (TLCSC 2) is to interface the AGSSS to all physical input and output devices. The DEVICE INTERFACE consists of 13 Low-Level Computer Software Components (LLCSCs):

1) Device Interface Executive (LLCSC 2.1),
2) Configuration Manager (LLCSC 2.2),
3) PC Front End (LLCSC 2.3),
4) PDG Support (LLCSC 2.4),
5) Terminal Interface (LLCSC 2.5),
6) Tablet Interface (LLCSC 2.6),
7) PC Interface (LLCSC 2.7),
8) PDG Interface (LLCSC 2.8),
9) File Interface (LLCSC 2.9),
10) Text Manager (LLCSC 2.10),
11) Graphics Manager (LLCSC 2.11),
12) File Router (LLCSC 2.12), and
13) Input Router (LLCSC 2.13).

The function of the DISPLAY EDITOR (TLCSC 3) is to support the generation of display formats from pictorial definition through action specifications and testing to display program integration. The DISPLAY EDITOR consists of 6 Low-Level Computer Software Components (LLCSCs):

1) AGSSS Executive (LLCSC 3.1),
2) Display Editor Menu (LLCSC 3.2),
3) Graphics Editor (LLCSC 3.3),
4) Actions Editor (LLCSC 3.4),
5) Display Test Manager (LLCSC 3.5), and
6) Display Program Generator (LLCSC 3.6).

The function of the RUN-TIME SUPPORT module (TLCSC 4) is to provide building blocks for use with display programs generated by AGSSS, in either of two modes:

1) In a stand-alone simulation called the Display Demonstrator, running on the same hardware as the AGSSS workstation, or

2) As part of an airborne software system (simulated or real).

The RUN-TIME SUPPORT module consists of two Low-Level Computer Software Components (LLCSCs):

1) Display Demonstrator Support (LLCSC 4.1), and

2) Mission Support (LLCSC 4.2).

The function of the AGSSS TOOLS (TLCSC 5) is to provide support packages of a general nature to the other TLCSCs and their corresponding LLCSCs and Units. The AGSSS TOOLS consists of 13 Low-Level Computer Software Components (LLCSCs):

1) Strings (LLCSC 5.1),
2) Sets (LLCSC 5.2),
3) Double Lists (LLCSC 5.3),
4) Trees (LLCSC 5.4),
5) Queues (LLCSC 5.5),
6) I/O Instantiations (LLCSC 5.6),
7) Miscellaneous Tools (LLCSC 5.7),
8) Storage Pool (LLCSC 5.8),
9) Generic Scanner I/O (LLCSC 5.9),
10) Convert PHIGS Archive Format (LLCSC 5.10),
11) DCL Interface (LLCSC 5.11),
12) Vectors (LLCSC 5.12),
13) Simulation Tools (LLCSC 5.13).

The overall decomposition of the AGSSS CSCI is presented in Figure 2.10. Note that the decomposition of the TLCSCs into LLCSCs and Units are identified only by number. The name of each LLCSC is included in Figure 2.11. Also note that the number of Units for each LLCSC is identified by the number inside the LLCSC box. The name of each Unit is also included in Figure 2.11.

The structure and organization of the AGSSS CSCI 1 of the AGSSS system is described in the AGSSS Software Detailed Design Document (SDDD) (see Ref. 7). It describes the decomposition of the AGSSS CSCI 1 into five TLCSCs and their corresponding Lower Level Computer Software Components and Units. In addition, this document defines the interface, data, and processing characteristics for each TLCSC, LLCSC, and Units in the AGSSS CSCI 1 design.

2.3 System Usage

The user interacts with the AGSSS system through various input and output devices. For input, physical devices include a keyboard and a pointing device, which may be either a mouse or a data tablet. A

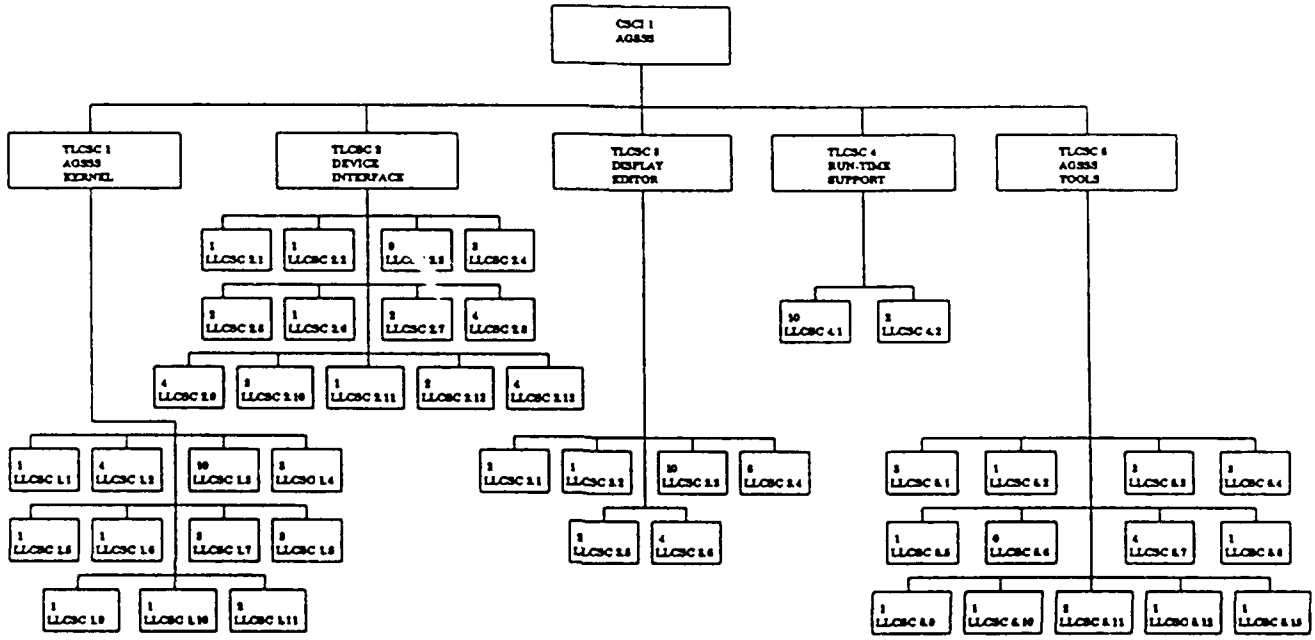


Figure 2.10 CSCI 1 General Decomposition of the AGSSS

TLCSC 1 AGSSS KERNEL

LLCSC 1.1 AGSSS EXECUTIVE

UNIT 1.1.1 AGSSS EXECUTIVE

LLCSC 1.2 WORKSPACE MANAGER

UNIT 1.2.1 WORKSPACE MANAGER INPUT MODULE
UNIT 1.2.2 WORKSPACE MANAGER MENU
UNIT 1.2.3 WORKSPACE MANAGER LIBRARY
UNIT 1.2.4 WORKSPACE MANAGER TASK

LLCSC 1.3 PHIGS MODULE

UNIT 1.3.1 STANDARD PHIGS
UNIT 1.3.2 PHIGS INPUT MODULE
UNIT 1.3.3 PHIGS GUARD TASK
UNIT 1.3.4 WORKSTATION EVENT QUEUE
UNIT 1.3.5 PHIGS NAME EXTENSIONS
UNIT 1.3.6 PHIGS PICK EXTENSIONS
UNIT 1.3.7 PHIGS MISCELLANEOUS EXTENSIONS
UNIT 1.3.8 PHIGS ARCHIVE FILE PARSER
UNIT 1.3.9 PHIGS WORKSTATION DESCRIPTION TABLE PARSER
UNIT 1.3.10 PHIGS EVENT QUEUE MANAGER

LLCSC 1.4 PATTI MODULE

UNIT 1.4.1 PATTI
UNIT 1.4.2 PATTI INPUT MODULE
UNIT 1.4.3 PATTI EVENT QUEUES

LLCSC 1.5 MENU MANAGER

UNIT 1.5.1 MENU MANAGER

LLCSC 1.6 FORMS MANAGER

UNIT 1.6.1 FORMS MANAGER

LLCSC 1.7 COLOR EDITOR

UNIT 1.7.1 COLOR EDITOR LIBRARY
UNIT 1.7.2 COLOR EDITOR MENU
UNIT 1.7.3 COLOR MANIPULATOR

LLCSC 1.8 FILE MANAGER

UNIT 1.8.1 FILE MANAGER LIBRARY
UNIT 1.8.2 FILE MANAGER I/O PACKAGES
UNIT 1.8.3 CUSTOM TEXT IO

LLCSC 1.9 MESSAGE LOGGER

UNIT 1.9.1 MESSAGE LOGGER

Figure 2.11 CSCI 1 Decomposition Elements of the AGSSS

LLCSC 1.10 DIALOG

UNIT 1.10.1 DIALOG

LLCSC 1.11 TEXT EDITOR

UNIT 1.11.1 INPUT HANDLER
UNIT 1.11.2 EDITOR LIBRARY

TLCSC 2 DEVICE INTERFACE

LLCSC 2.1 DEVICE INTERFACE EXECUTIVE

UNIT 2.1.1 IFACE_EXEC

LLCSC 2.2 CONFIGURATION MANAGER

UNIT 2.2.1 CONFIGURATION MANAGER

LLCSC 2.3 PC FRONT END

UNIT 2.3.1 PC FRONT END MAIN
UNIT 2.3.2 SCHEDULER
UNIT 2.3.3 HOST INTERFACE
UNIT 2.3.4 DISK I/O MANAGER
UNIT 2.3.5 DISPLAY MANAGER
UNIT 2.3.6 KEYBOARD MANAGER
UNIT 2.3.7 MOUSE MANAGER
UNIT 2.3.8 VOICE MANAGER
UNIT 2.3.9 LOW LEVEL ROUTINES

LLCSC 2.4 PDG SUPPORT

UNIT 2.4.1 PDG GRAPHICS PROGRAM
UNIT 2.4.2 PDG MESSAGE QUEUE
UNIT 2.4.3 ADAGE LIBRARY

LLCSC 2.5 TERMINAL INTERFACE

UNIT 2.5.1 TERMINAL INTERFACE INPUT TASK
UNIT 2.5.2 DISPLAY_TASK

LLCSC 2.6 TABLET INTERFACE

UNIT 2.6.1 TABLET INTERFACE

LLCSC 2.7 PC INTERFACE

UNIT 2.7.1 PC MONITOR
UNIT 2.7.2 PC SERVICES

LLCSC 2.8 PDG INTERFACE

UNIT 2.8.1 PDG OUTPUT TASK

Figure 2.11 (continued)

UNIT 2.8.2 PDG PHIGS ECHO TASK
UNIT 2.8.3 PDG SERVER QUEUES
UNIT 2.8.4 PDG SERVER

LLCSC 2.9 FILE INTERFACE

UNIT 2.9.1 FILE TYPES
UNIT 2.9.2 FILE INTERFACE LIBRARY
UNIT 2.9.3 FILE INTERFACE I/O PACKAGES
UNIT 2.9.4 SEARCH LIBRARY

LLCSC 2.10 TEXT MANAGER

UNIT 2.10.1 PATTI OUTPUT TASK
UNIT 2.10.2 ECHO TASK
UNIT 2.10.3 PICK RETURNS TASK

LLCSC 2.11 GRAPHICS MANAGER

UNIT 2.11.1 GRAPHICS MANAGER

LLCSC 2.12 FILE ROUTER

UNIT 2.12.1 FILE ROUTER LIBRARY
UNIT 2.12.2 FILE ROUTER I/O PACKAGES

LLCSC 2.13 INPUT ROUTER

UNIT 2.13.1 INPUT ROUTER LOGICAL INPUTS LIBRARY
UNIT 2.13.2 INPUT ROUTER INPUT HANDLER
UNIT 2.13.3 INPUT ROUTER ECHO HANDLER
UNIT 2.13.4 FINITE STATE MACHINES

TLCSC 3 DISPLAY EDITOR

LLCSC 3.1 APPLICATION EXECUTIVE

UNIT 3.1.1 AGSSS MAIN PROGRAM
UNIT 3.1.2 APPLICATION EXECUTIVE TASK

LLCSC 3.2 DISPLAY EDITOR MENU

UNIT 3.2.1 DISPLAY EDITOR MENU

LLCSC 3.3 GRAPHICS EDITOR

UNIT 3.3.1 GRAPHICS EDITOR MENU
UNIT 3.3.2 DISPLAYABLE STORES MENUS
UNIT 3.3.3 DISPLAYABLE STORE MENUS
UNIT 3.3.4 STRUCTURE MENUS
UNIT 3.3.5 ELEMENT CREATION
UNIT 3.3.6 FILE STORES MENUS
UNIT 3.3.7 FILE STORE MENUS
UNIT 3.3.8 PHIGS WORKSTATION MANAGER MENU

Figure 2.11 (continued)

UNIT 3.3.9 PHIGS WORKSTATION STATE LIST UTILITIES
UNIT 3.3.10 GRAPHICS EDITOR UTILITIES

LLCSC 3.4 ACTIONS EDITOR

UNIT 3.4.1 ACTIONS EDITOR EXECUTIVE
UNIT 3.4.2 SOFTWARE LIBRARY MANAGER
UNIT 3.4.3 DIANA MANAGER
UNIT 3.4.4 SYNTAX-DIRECTED EDITOR
UNIT 3.4.5 COMPILER FRONT END
UNIT 3.4.6 INTERPRETER
UNIT 3.4.7 LANGUAGE TOOLS
UNIT 3.4.8 ADAGE 3000 CROSS-COMPILER BACK END

LLCSC 3.5 DISPLAY TEST MANAGER

UNIT 3.5.1 DISPLAY TEST EXECUTIVE
UNIT 3.5.2 DISPLAY TEST CONFIGURATION MANAGER

LLCSC 3.6 DISPLAY PROGRAM GENERATOR

UNIT 3.6.1 DISPLAY PROGRAM GENERATOR MENU
UNIT 3.6.2 DISPLAY GENERATOR
UNIT 3.6.3 DISPLAY RESTORER
UNIT 3.6.4 COMMAND FILE GENERATOR

TLCSC 4 RUN-TIME SUPPORT

LLCSC 4.1 DISPLAY DEMONSTRATOR SUPPORT MODULE

UNIT 4.1.1 SIMULATOR EXECUTIVE
UNIT 4.1.2 SIMULATOR MONITOR
UNIT 4.1.3 SIMULATOR PRECISION CLOCK
UNIT 4.1.4 SIMULATOR INPUT MODULE
UNIT 4.1.5 SIMULATOR AIRPLANE
UNIT 4.1.6 SIMULATOR FLIGHT RECORDER
UNIT 4.1.7 SIMULATOR DISPLAY SYSTEM
UNIT 4.1.8 SIMULATOR INPUT DATA GUARD
UNIT 4.1.9 SIMULATOR FLIGHT DATA GUARD
UNIT 4.1.10 SIMULATOR DATA GUARD

LLCSC 4.2 MISSION SUPPORT

UNIT 4.2.1 MISSION EXECUTIVE
UNIT 4.2.2 MISSION AVIONICS MODULE

TLCSC 5 AGSSS TOOLS

LLCSC 5.1 STRINGS

UNIT 5.1.1 VAR_STRING LIBRARY
UNIT 5.1.2 VAR_STRING INPUT/OUTPUT LIBRARY
UNIT 5.1.3 STRING SCANNER

Figure 2.11 (continued)

LLCSC 5.2 SETS

UNIT 5.2.1 SETS

LLCSC 5.3 LISTS

UNIT 5.3.1 SINGLE LISTS
UNIT 5.3.2 DOUBLE LISTS
UNIT 5.3.3 ORDERED LISTS

LLCSC 5.4 TREES

UNIT 5.4.1 BINARY TREES
UNIT 5.4.2 IPR TREES
UNIT 5.4.3 N-ARY TREES

LLCSC 5.5 QUEUES

UNIT 5.5.1 QUEUES

LLCSC 5.6 I/O INSTANTIATIONS

LLCSC 5.7 MISCELLANEOUS TOOLS

UNIT 5.7.1 BIT-WISE OPERATIONS
UNIT 5.7.2 MATH LIBRARY EXTENSIONS
UNIT 5.7.3 TIME AND DATE ROUTINES
UNIT 5.7.4 IMMEDIATE-IF ROUTINES

LLCSC 5.8 STORAGE POOL

UNIT 5.8.1 STORAGE POOL

LLCSC 5.9 GENERIC SCANNER IO

UNIT 5.9.1 GENERIC SCANNER IO

LLCSC 5.10 CONVERT PHIGS ARCHIVE FORMAT

UNIT 5.10.1 CONVERT PHIGS ARCHIVE FORMAT

LLCSC 5.11 DCL INTERFACE

UNIT 5.11.1 DCL SPAWN
UNIT 5.11.2 COMMAND LINE INTERFACE

LLCSC 5.12 VECTORS

UNIT 5.12.1 VECTORS

LLCSC 5.13 SIMULATION TOOLS

UNIT 5.13.1 SIMULATION TOOLS

Figure 2.11 (concluded)

keyboard-based mouse emulator may be used as an option. This may be supplemented by the voice input module on the PC Front End. The AGSSS uses one or more display screens for output. One of these screens is the alphanumeric terminal through which the user has logged onto the host computer system. Another screen may be that of the PDG which must be available for the viewing of graphics. In addition, any number of additional alphanumeric terminals may be used simultaneously including that of the PC Front End. Text-only workspaces may be located on any of these, while graphics workspaces may only reside on the PDG. A combination of microcode running on the PDG and host-based software implements PHIGS for displaying the graphics.

Each of these output display screens is referred to as a metastation. A metastation may be one of two types: text-only or text/graphics. These two types are referred to as PATTI and PHIGS metastations, respectively, in reference to the text-only and graphics interface standards used by the AGSSS internal software components. (PATTI is the Programmable Attributed Text Interface, while PHIGS is the Programmer's Hierarchical Interactive Graphics System.)

Underlying the AGSSS is a window-based graphical user interface and higher-level tools such as standard menu and form interaction modules. The individual components of the AGSSS, such as the Actions Editor, Graphics Editor, etc., are the clients or applications of this underlying system. The windowing system is two-tiered, consisting of workspaces and workstations. The workspace is the higher-level window, or rectangular region of a screen, which an application must first open to communicate with the user. Within a workspace, one or more workstations can then be opened. These workstations are windows within the workspace through which the actual input and output occur. Thus a workspace can be a region of the screen within which a given application operates, and binds together a collection of workstations owned by that application. The boundary of a workspace also serves as a clipping boundary, keeping the output from multiple applications separate on the screen.

For graphical input/output, the AGSSS workstation concept is mapped directly onto the PHIGS definition of a workstation. While the PHIGS standard does not specifically detail the possibility of implementing a workstation in such a way that more than one application can utilize the same physical hardware simultaneously, it does not rule out such a possibility either. Thus some applications within AGSSS open multiple PHIGS workstations on the PDG. These may be "held together" by a common workspace, which the user is free to manipulate (size, move, etc.) on the screen using the Workspace Manager utility. In addition, the PHIGS workstation concept is extended even farther in AGSSS through RTI's concurrent PHIGS implementation which allows multiple, independent applications to control their own workstations on different workspaces on the same display hardware.

Of all the workstations open at any given time, one is designated as the listener workstation. The workspace which owns it is called the listener workspace, and the metastation on which that workspace is located is called the listener metastation. All input is directed to the listener workstation or, more precisely, to the application which owns it. Workstations and workspaces may be moved about and changed in size by the use of the Workspace Manager. The windows are allowed to overlap each other, according to a viewing priority which may be modified either by pushing them from the Workspace manager or by changing the listener.

Menus are the primary means of communicating choice information to the system (applications). A standard menu interface which is layered on top of the lower-level text I/O capabilities of PATTI is utilized throughout the AGSSS for such operations. All menus using this interface thus have a common "look and feel." An application may have more than one menu displayed simultaneously and, in addition, more than one of these may be "live" -- that is, ready to accept a choice. Examples of an AGSSS menu is illustrated in Figure 2.12. The Display Editor Menu, the Actions Editor Menu, and the Color Editor Menu shown in this figure is the top-most menu of the system, and it is used to provide access to all other parts of AGSSS.

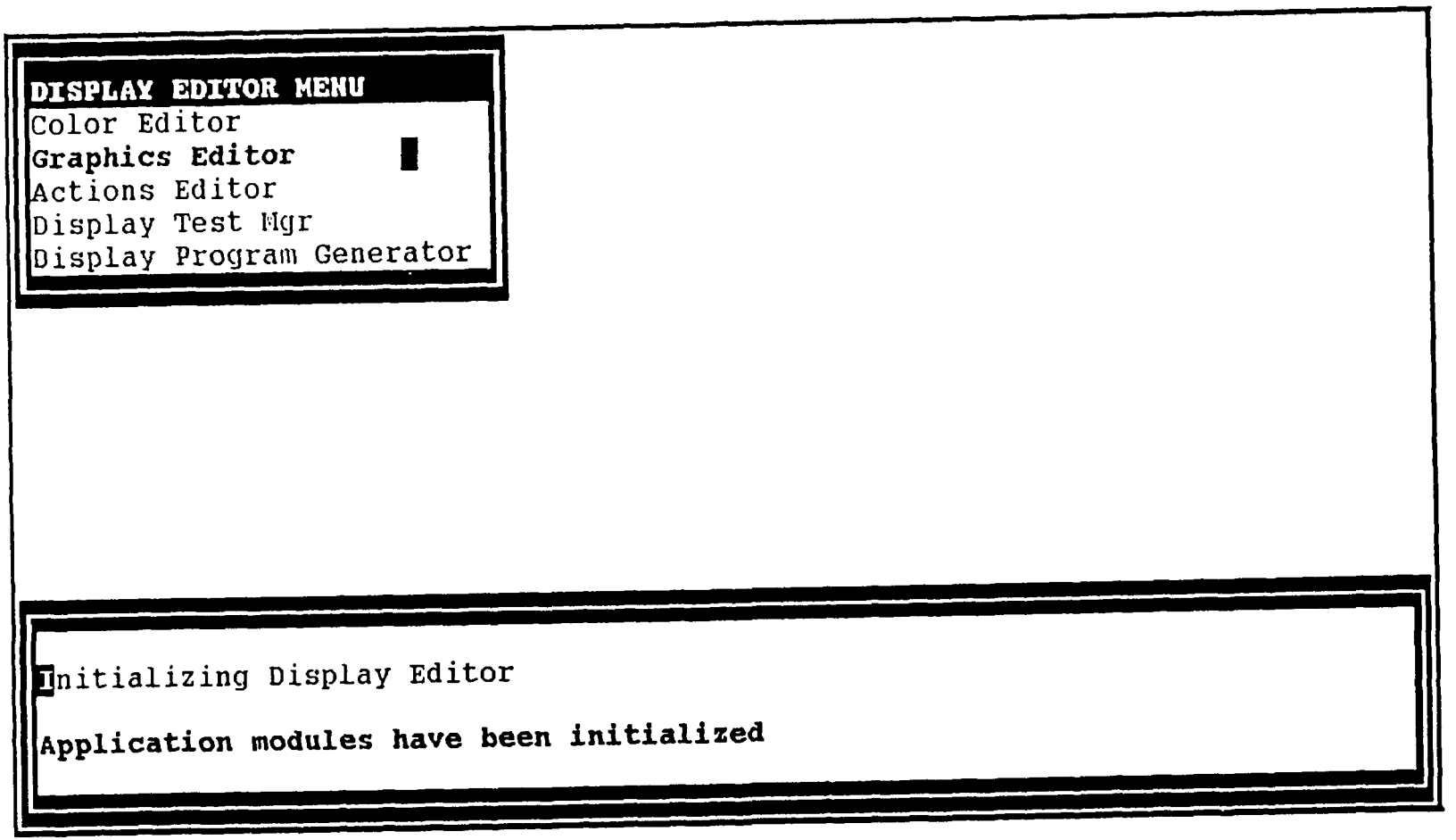


Figure 2.12 Example of an AGSSS Menu: Display Editor Menu

A higher-level method for use by applications to input alphanumeric data from the user is available through the Forms module, which is layered on top of PATTI. A form is simply a collection of one or more data input fields with associated prompt text and data validation rules. Data may be typed or edited in the separate fields of a form in any order. Clicking on a [DONE] button will have the effect of transmitting all the data to the application as it appears at that time. Some forms have on-line help available indicated by the presence of a [HELP] button. In addition, forms which perform validity tests on the data provide a button labelled "[ERROR]" which will flash if an invalid value has been entered in a field. Clicking on this button while it is flashing will bring up an explanation of the error. Figure 2.13 depicts a form used in conjunction with the flight simulator during display test procedures.

For simple input of a single data item at a time, some AGSSS components use a higher-level module similar in many ways to a form, and also layered on top of PATTI, known as a dialog box. A dialog box is simply a "pop-up" query to be responded to by the user of AGSSS. Dialog boxes allow the user to provide input which may consist of a string or number, and which is completed by pressing the Return key, or may be cancelled by clicking on a [CANCEL] button. Another use of dialog boxes is to display a message and wait until it has been acknowledged. These boxes only have a single button labelled [OK], and go away when that button is clicked. Another form of dialog box is the Boolean query, in which a question is asked and buttons for [YES] and [NO] or for [TRUE] and [FALSE] are provided.

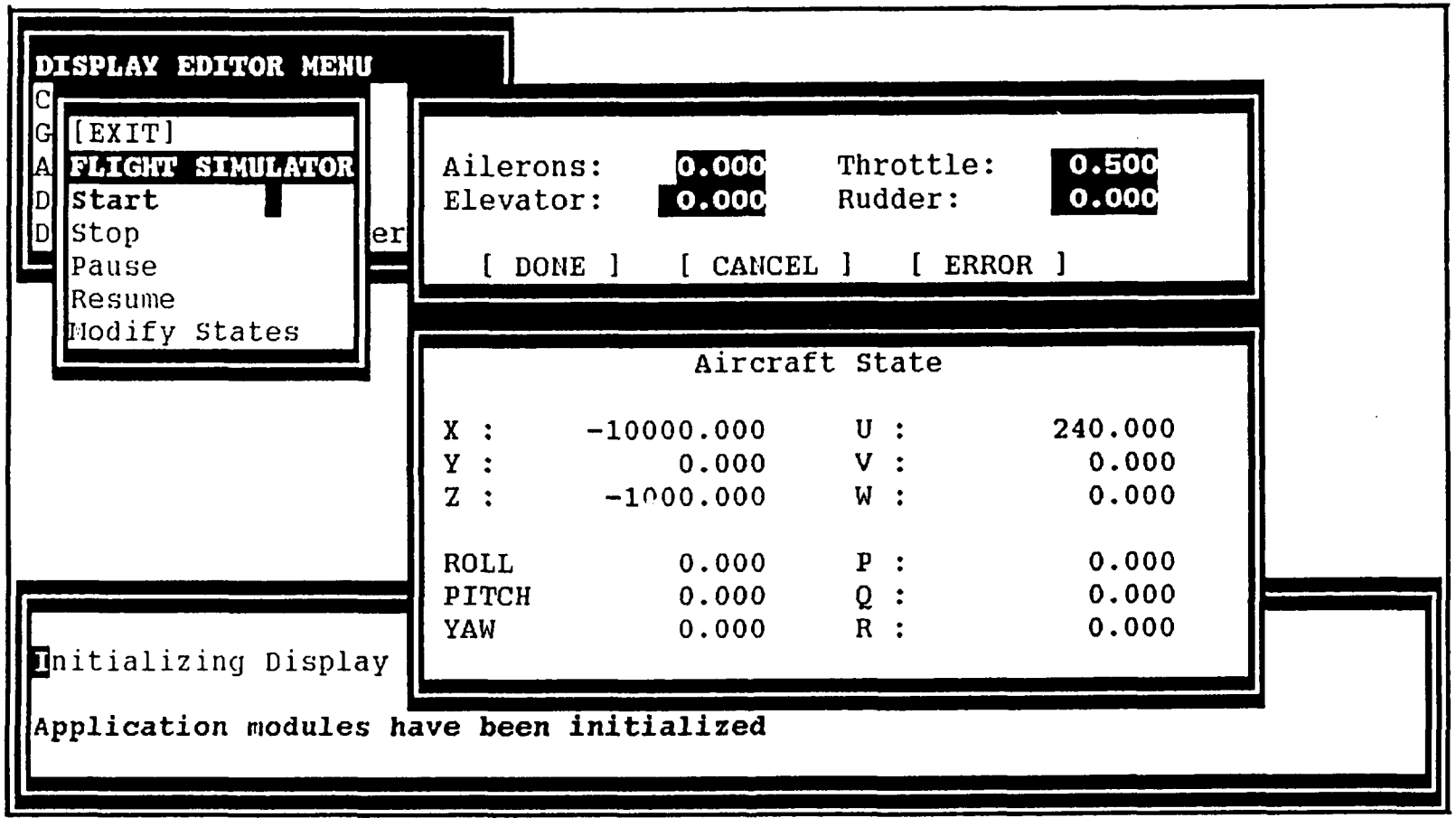


Figure 2.13 Example of an AGSSS Form: Flight Simulator Form

3.0 DISPLAY DEVELOPMENT USING AGSSS

This section presents a detailed description of the application of the AGSSS to the cockpit display development process. It also includes several examples of displays and code developed using the AGSSS.

Application of AGSSS to the development of a display format and its enabling software consists of the following four steps.

First, static graphics objects or pictures are generated. This is done through the Graphics Editor, and may utilize graphics imported from other PHIGS-based systems through the use of the PHIGS standard archive file format. The Graphics Editor allows the user to interactively create, edit, combine, delete, and transfer graphical objects described using the hierarchical structure-based concepts specified by the PHIGS standard. The Graphics Editor maintains two types of PHIGS structure stores in system memory: the displayable stores and the file stores. The displayable stores are created by the user to generate the graphics seen on the display. The file stores are the internal representation of the structures in archive files. Multiple stores of each kind are possible, and are distinguished by user-specifiable names.

Closely associated with the Graphics Editor is the Color Editor which is a utility application available to allow the user to graphically specify a color, with immediate feedback as to its appearance, using any of three different color models: RGB, HSV, or CIE. The selected color may then be used by the Graphics Editor for specifying a color attribute.

This portion of the display development process is illustrated in Figures 3.1 through 3.6. Figure 3.1 shows the main AGSSS menu (Display Editor Menu) and its use to invoke the Color Editor in a PATTI workstations. Figure 3.2 shows the Graphics Editor being used to

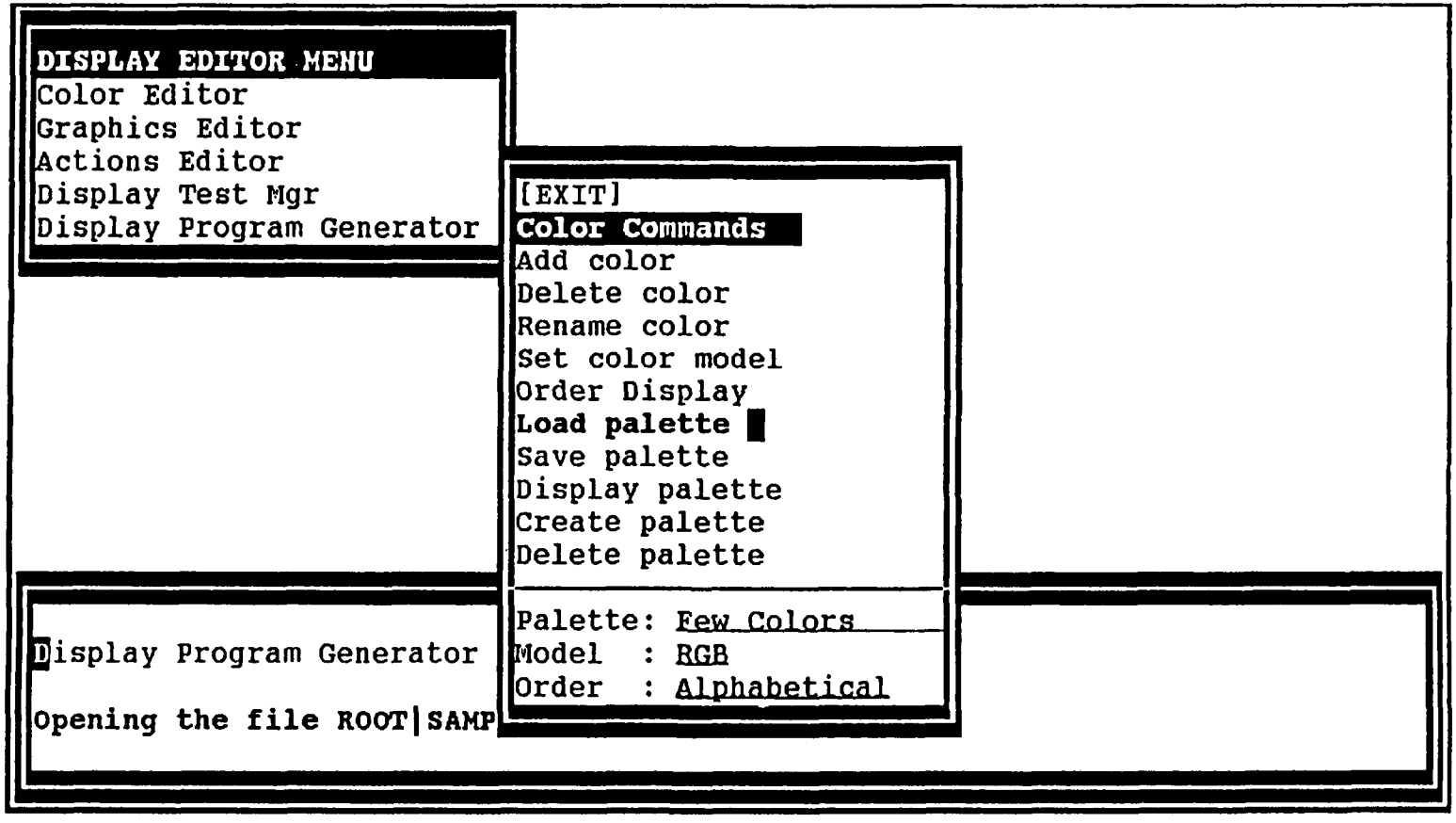


Figure 3.1 Use of the Graphics Menu to Invoke the Color Editor Menu

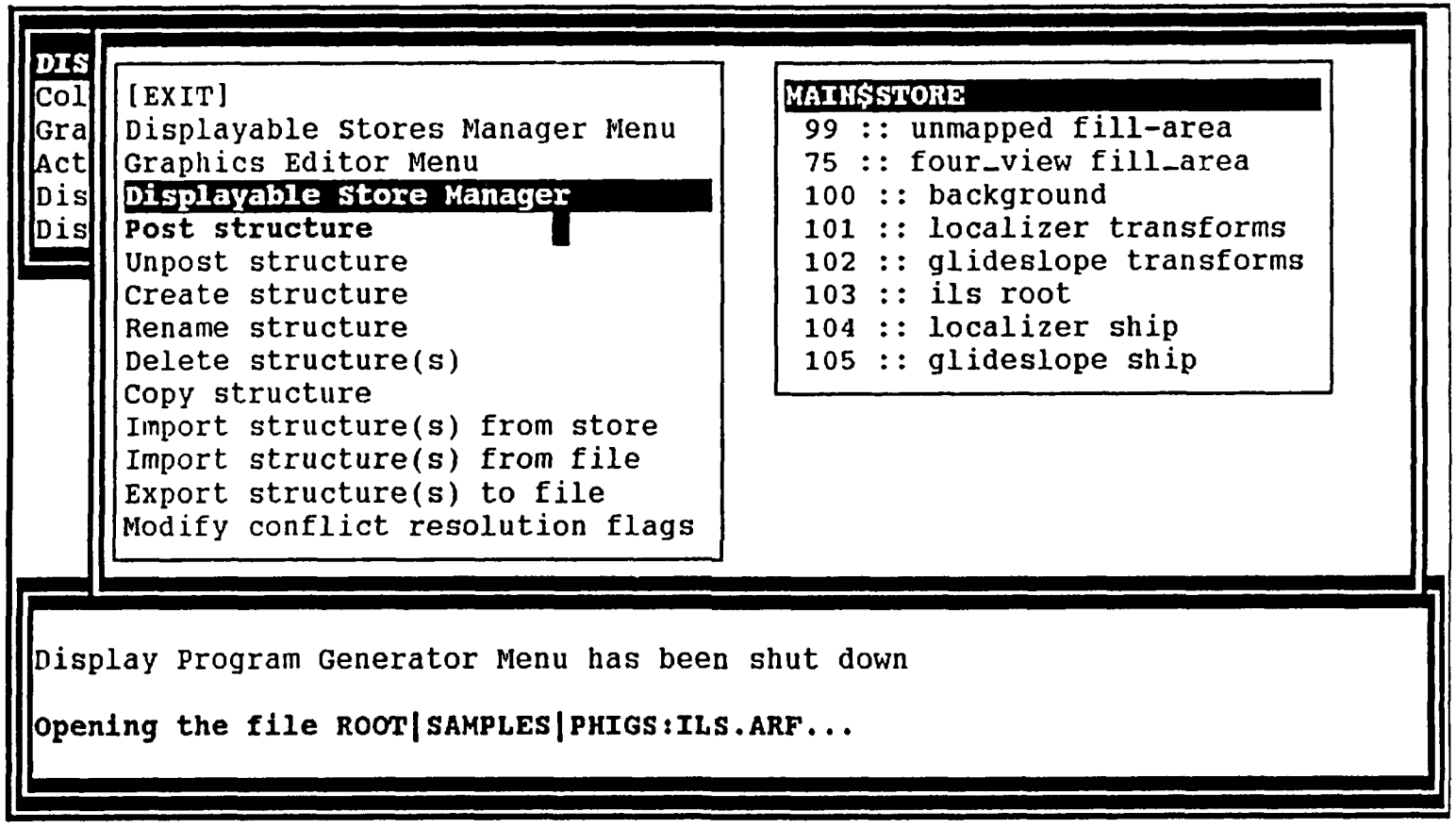


Figure 3.2 Use of the Graphics Editor to Traverse Displayable PHIGS Stores

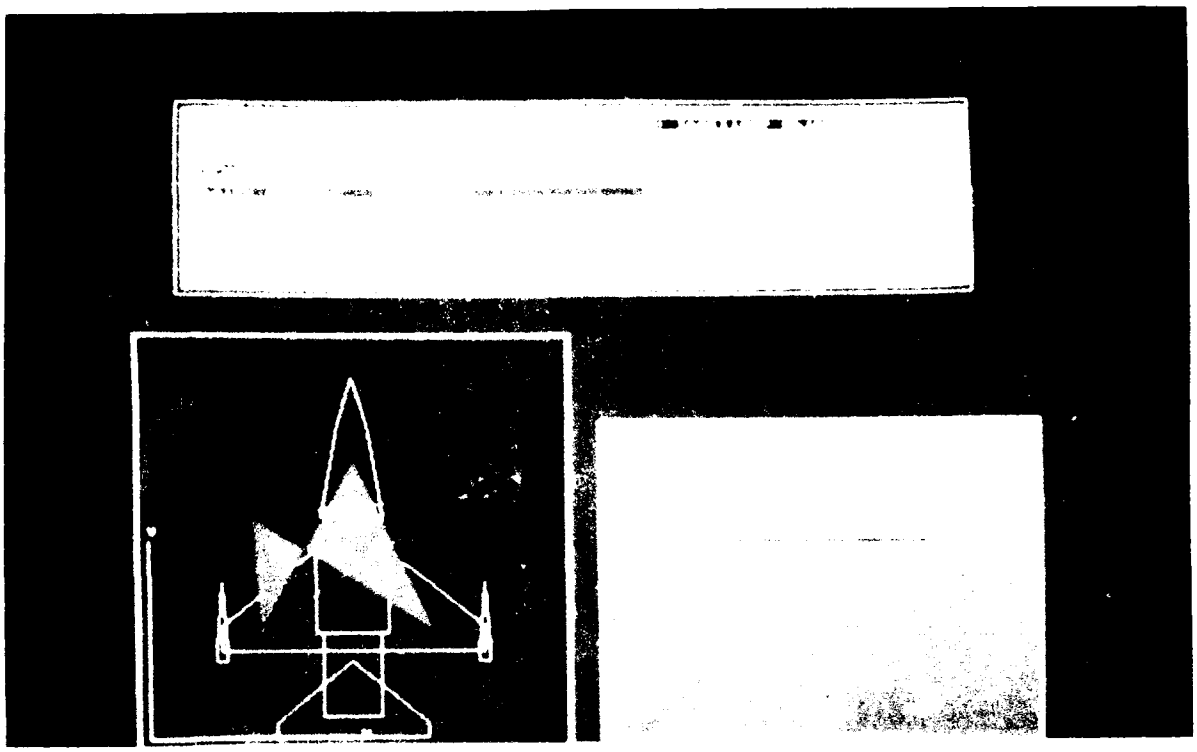


Figure 3.3 Use of Graphics Editor to Add (Draw) a Feature (Star) to an Existing (Aircraft) Display

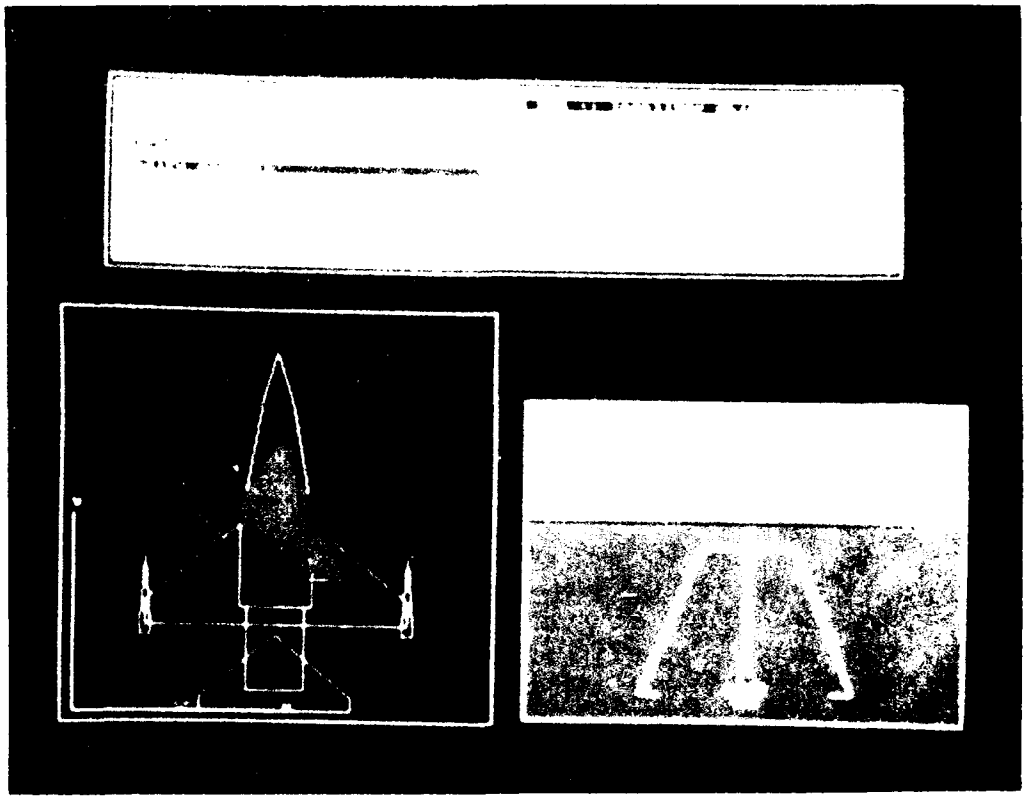
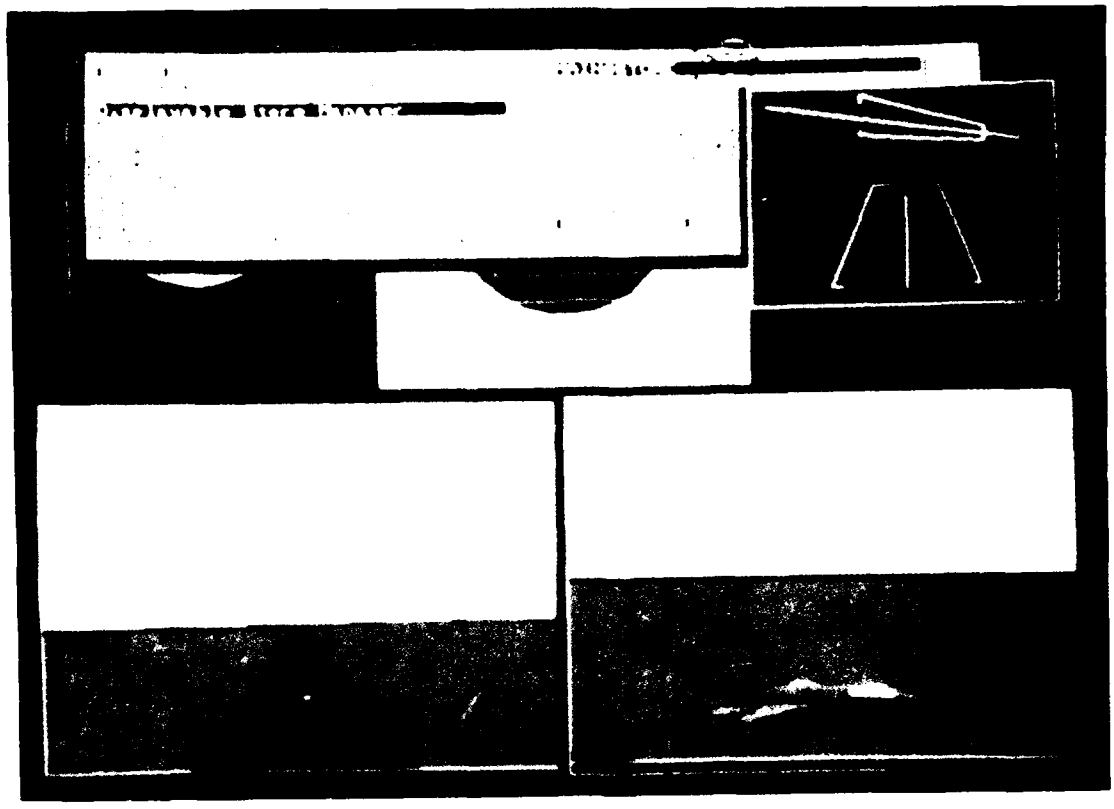


Figure 3.4 Use of Graphics Editor to Complete (Hide) the Drawing of the Star



3.5 Example of Several Display Formats
Instanced on Several PHIGS Workstations

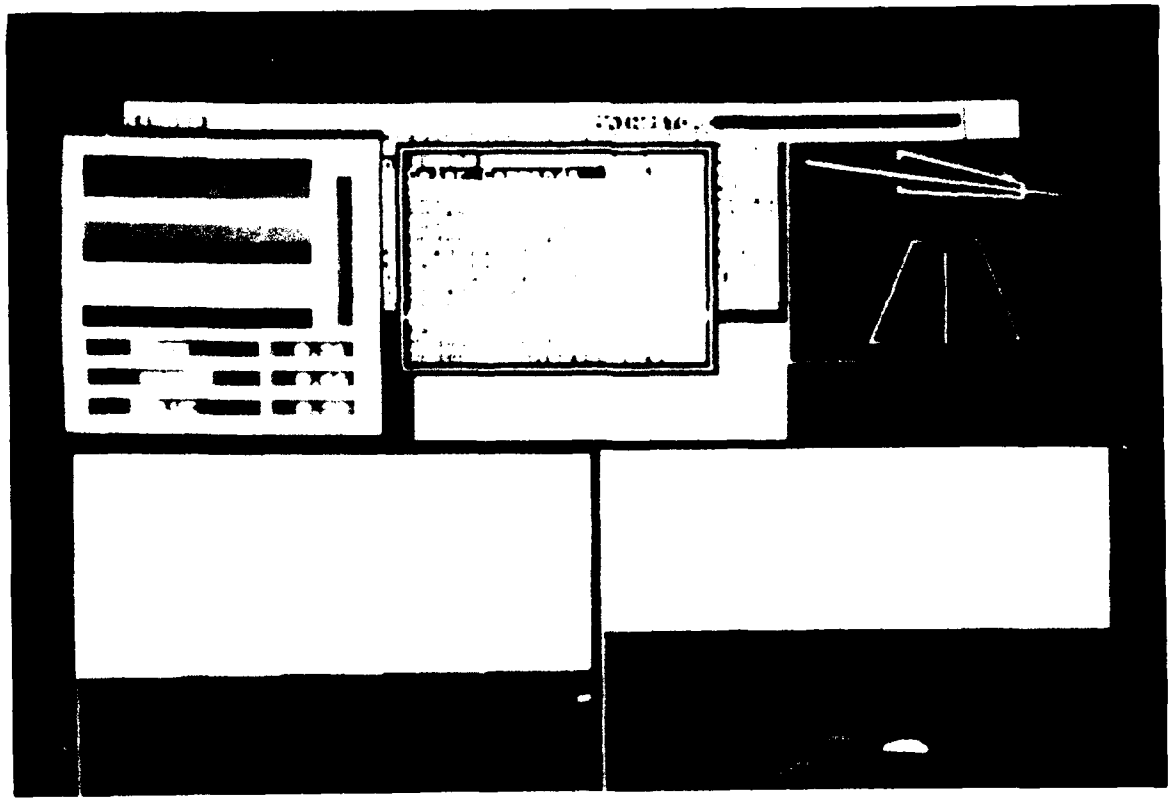


Figure 3.6 Example of Several Display Formats
and Use of the Color Editor With One of Them

traverse a particular displayable PHIGS stores. Figures 3.3 and 3.4 show the Graphics Editor in action supporting the drawing of the outline of a star to be added to an existing aircraft display. This sequence show the rubberbanding of the star in Figure 3.3 and the completion of the flat-shaded star in Figure 3.4. Figure 3.5 shows the versatility of the PHIGS structure concept as used by the AGSSS Graphics Editor. In this figure, the Graphics Editor has been used to bring up multiple display stores on several PHIGS workstations. Figure 3.6 is similar to Figure 3.5 except for the inclusion of the Color Editor. This can be used to edit the color of one of the displays on line.

Second, the rules for animating the graphics in response to specified system inputs are coded. This is done using the Actions Editor, which contains a syntax-directed editor and a parser and interpreter for a subset of Ada known as SOFA (Subset **of** Ada). The Actions Editor also contains predefined code modules to facilitate the development of display programs. The code thus generated refers to the structures in the PHIGS archive file(s) generated by the Graphics Editor, which are accessed through the PHIGS calls. During an Actions Editor session, communication with the Graphics Editor is utilized to obtain and display information on the graphics structures that the AGSSS knows about as a result of the user interaction with that application.

The Actions Editor allows the display designer to define, execute, and debug display programs within an integrated programming environment. The source language for the display action specification is SOFA, a strict subset of Ada. SOFA is basically Ada minus tasks, generics, and variant records. All of the components of the Actions Editor make use of an intermediate Ada representation called DIANA, which provides data structures for attributed Ada syntax trees.

The display designer can define programs by supplying parameters to predefined display actions (specified as subprograms in predefined compilation units), and/or may choose to create or edit source code directly. Both forms of display definition take place within a syntax-directed editor (SYNDE), which parses the code to check the syntactic

and semantic content of the source program as it is being created or modified. The predefined SOFA environment also includes the PHIGS package specification for making calls to PHIGS. In addition to assisting the display designer to create grammatically correct programs, the Syntax-Directed Editor uses the parser front end incrementally to generate an equivalent DIANA representation of the source program.

An interpreter component of the Actions Editor allows the program to be executed for debugging purposes. By making use of the DIANA internal form of display programs, the interpreter module allows the display designer to execute display programs without the traditional compilation/link phase. The Actions Editor can also be called upon to execute the current program by the Display Test Manager when the user wishes to more thoroughly test the program using that module.

The Actions Editor is integrated with the Graphics Editor for executing calls to the PHIGS module to interact with the graphics structures. It can also use the Color Editor to specify a color attribute.

The Actions Editor generates output in the form of program source files in the SOFA language. It will also respond to requests from the Display Program Generator for source code output, and from the Display Test Manager for interpreting the current display program.

This portion of the display development process is illustrated in Figures 3.7 through 3.10. Figure 3.7 shows the use of the Display Editor Menu to invoke the Actions Editor. Figure 3.8 shows additional menus and forms associated with the Actions Editor. These are shown on a separate workspace on the foreground of the metastation. The form shown here is used to create or recall a particular Ada action program into the editor. In this example the existing program ILS is being called back for modification/additions to it. The additional menus are used to specify desired actions on this program. Figure 3.9 shows a portion of the body of program ILS which has been invoked through the traversal of several menus and forms. Finally, Figure 3.10 shows the

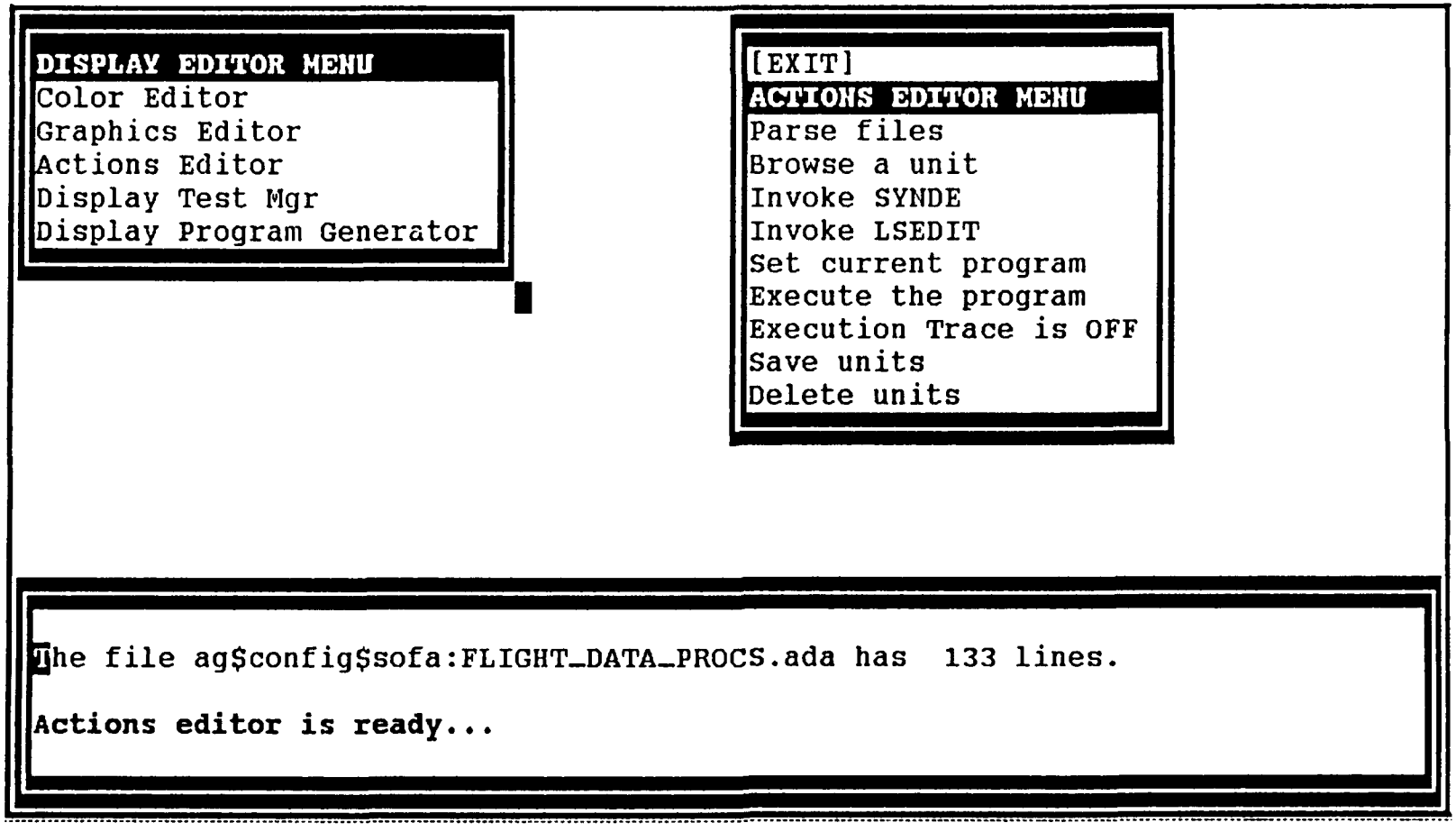


Figure 3.7 Use of the Display Editor Menu
to Invoke the Actions Editor Menu

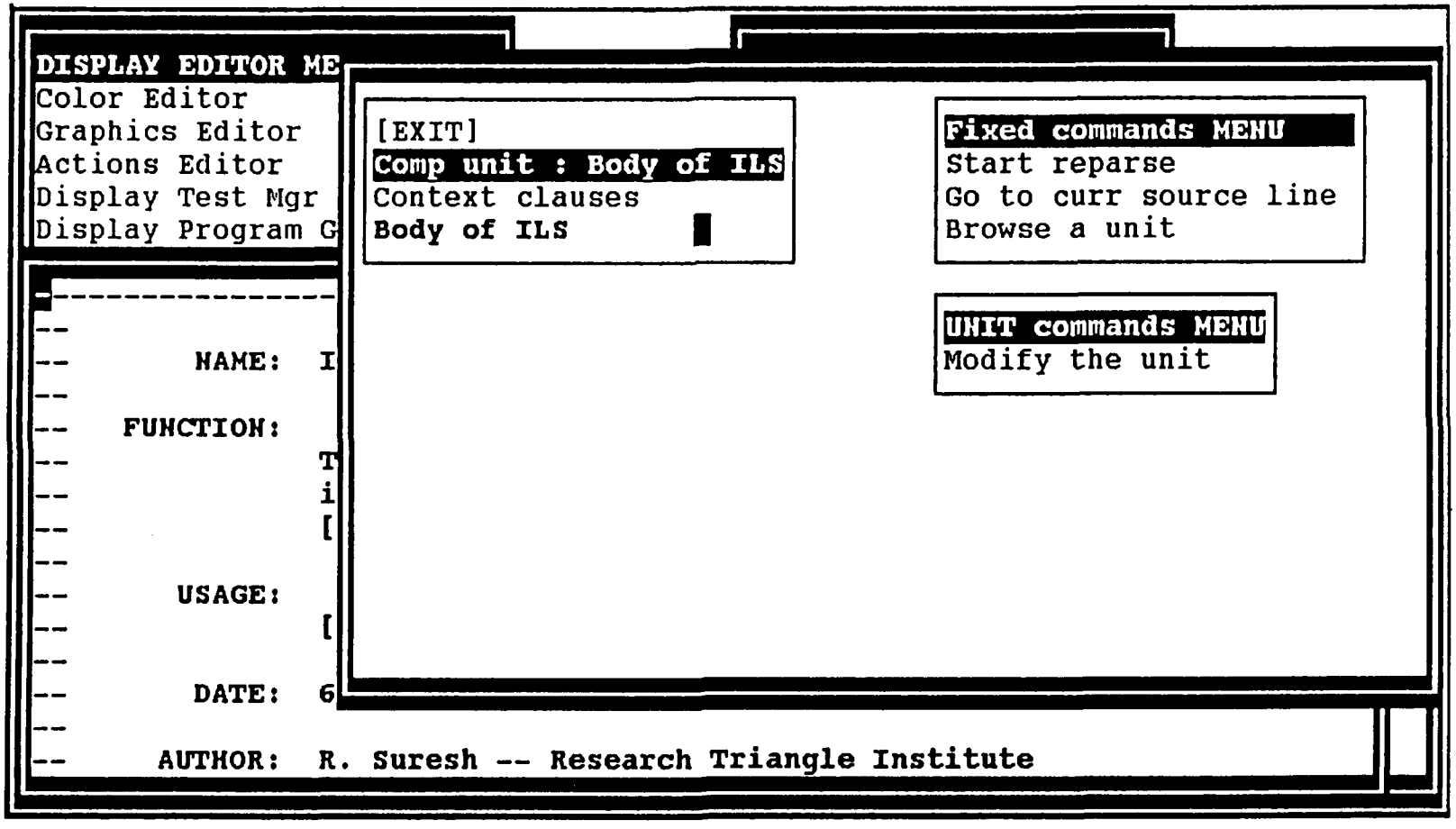


Figure 3.8 Example of Additional Menus and Forms Associated With the Actions Editor

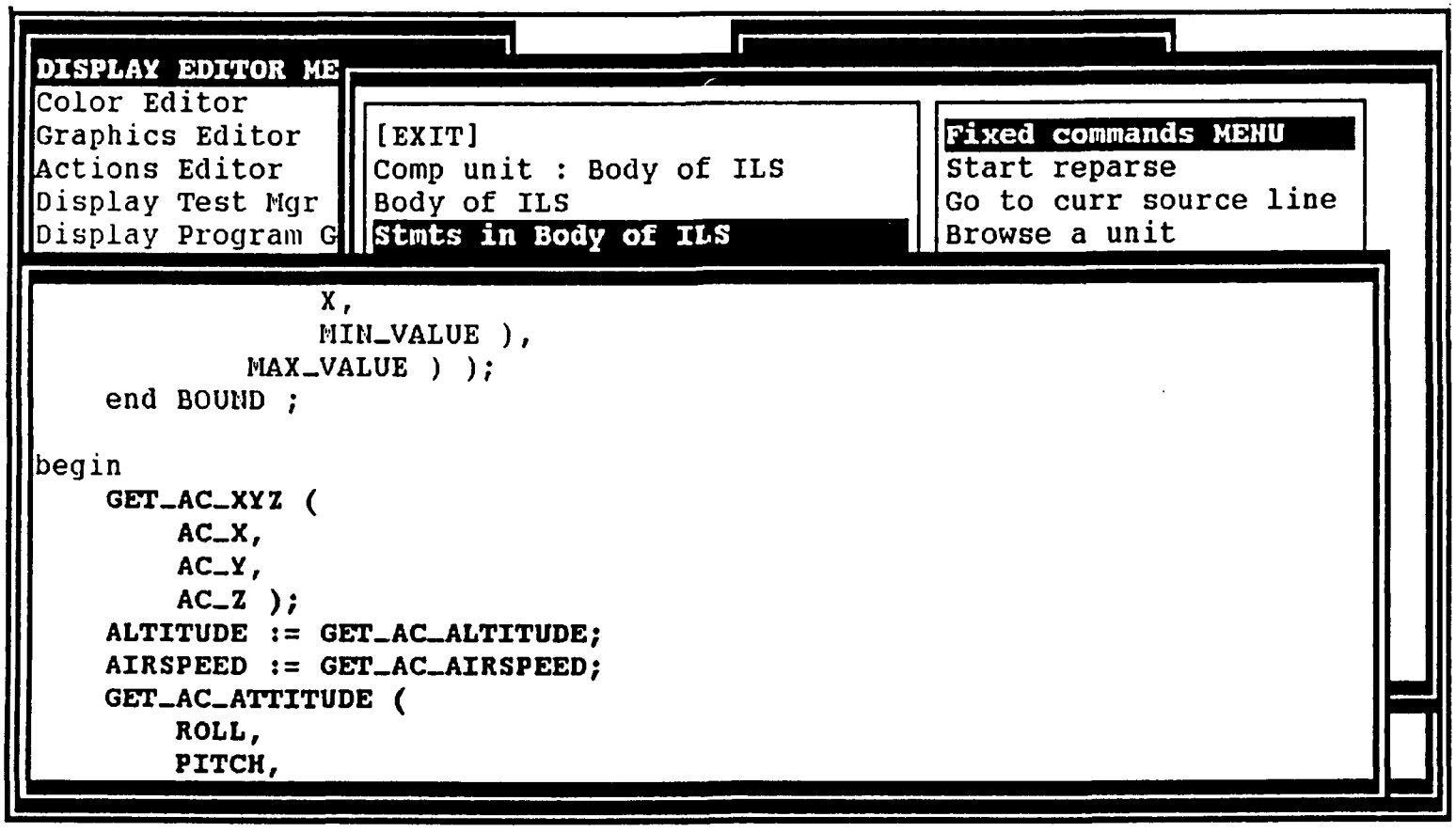


Figure 3.9 Portion of the Body of ILS

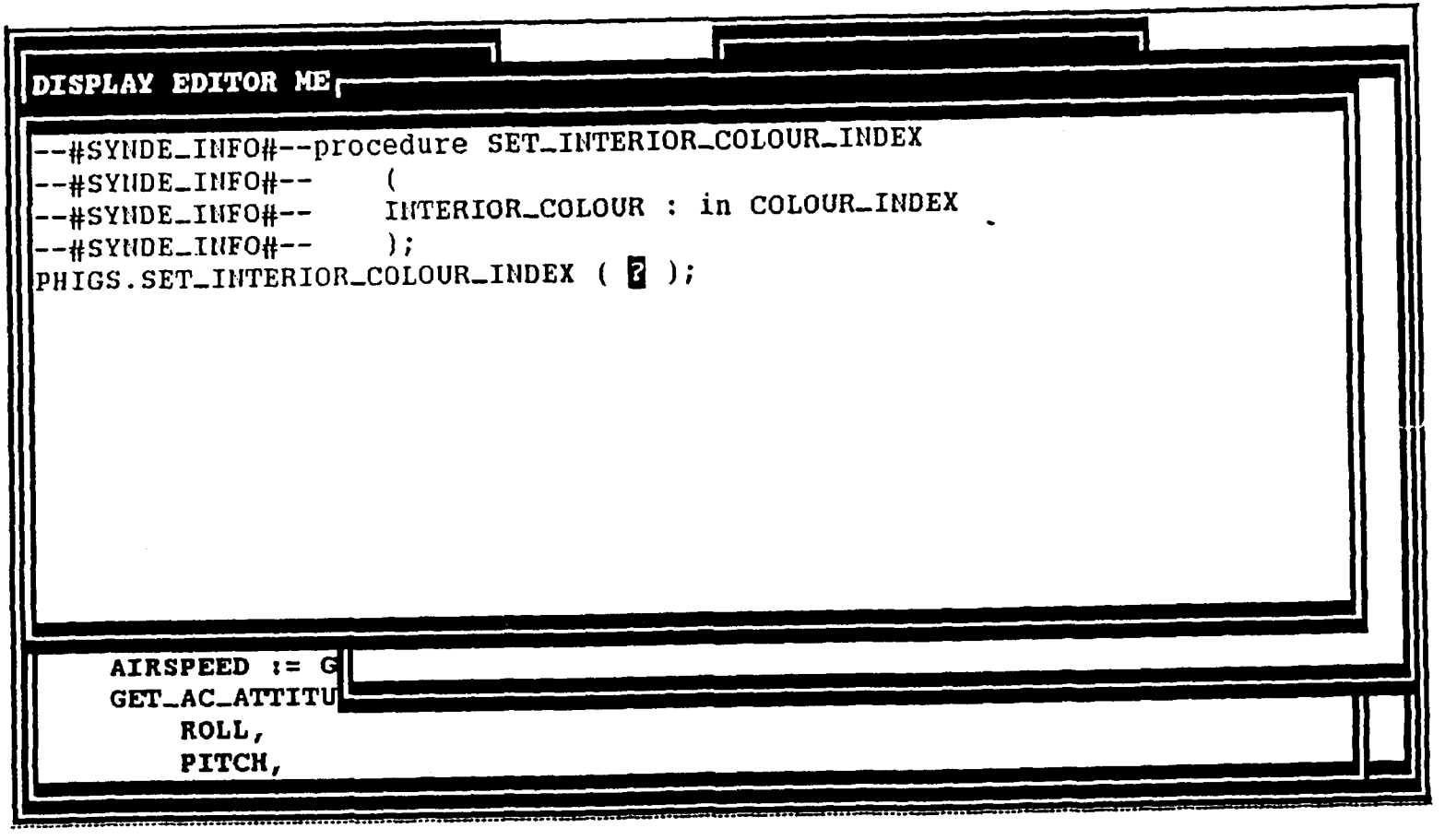


Figure 3.10 Example of Changes to the Body of ILS

mechanism through which the designer can modify an existing statement or add a statement to the program. This figure shows the use of SYNDE for this purpose.

Third, the display program and graphical data thus generated may be tested using the Display Test Manager, which enables the user to manipulate the inputs to the display program and observe their effects on the graphics display. This application utilizes the Actions Editor to run the current display program through its interpreter, and the Graphics Editor to access and manipulate the corresponding graphics structures. The Display Test Manager contains a flight simulator, the outputs of which can be fed as inputs to the display program currently in the Actions Editor for interpretation.

The Display Test Manager communicates with the Actions Editor application to initiate interpretation of the current display program, and communicates aircraft state data to it. In addition, it outputs the aircraft state data and the latest pilot inputs to the screen in a PATTI workstation.

This portion of the display development process is illustrated in Figures 3.11 through 3.14. Figure 3.11 shows a form used to control the "onboard" (within AGSSS) aircraft simulator that will animate the display under development through the Graphics Editor according to the rules outlined by the Actions Editor. This form is invoked through the Display Test Manager Option of the Display Editor Menu. Figures 3.12 through 3.14 show multiple displays associated with (being animated by) the aircraft simulation. Of particular interest in these three figures is the sequence of two views at the bottom. The scene on the left represent the out-the-window view from the cockpit of the aircraft whereas the scene on the right represent the view from the tower at the airport from which the aircraft is taking off. As the aircraft makes its take-off run, the scenes vary with the tower coming into view and then disappearing (on left) and the aircraft symbol increasing in size as it approaches and flies by the tower. These two displays represent different views of the same "data base" and illustrate the flexibility

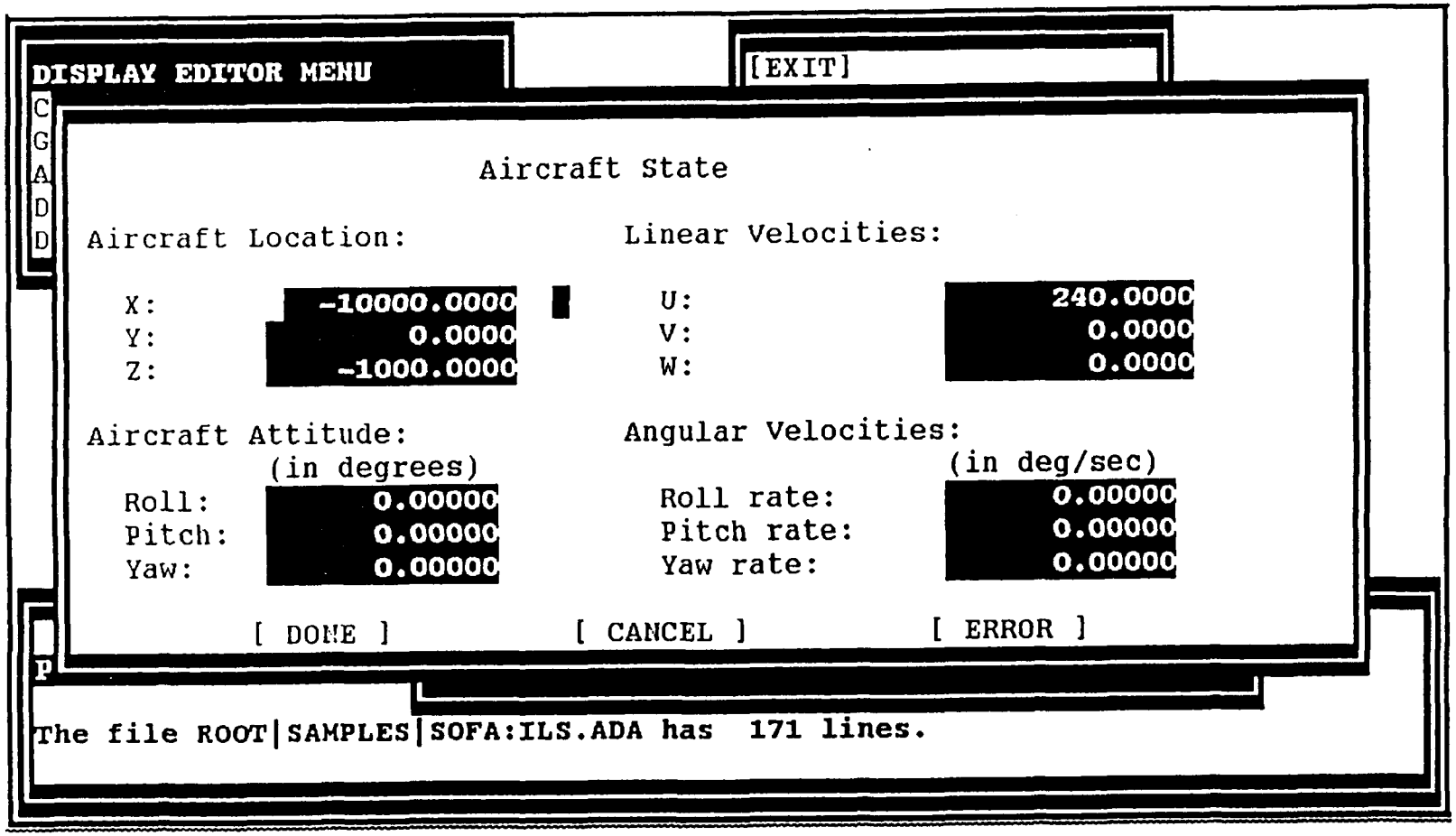


Figure 3.11 AGSSS Onboard Aircraft Simulator Control Form

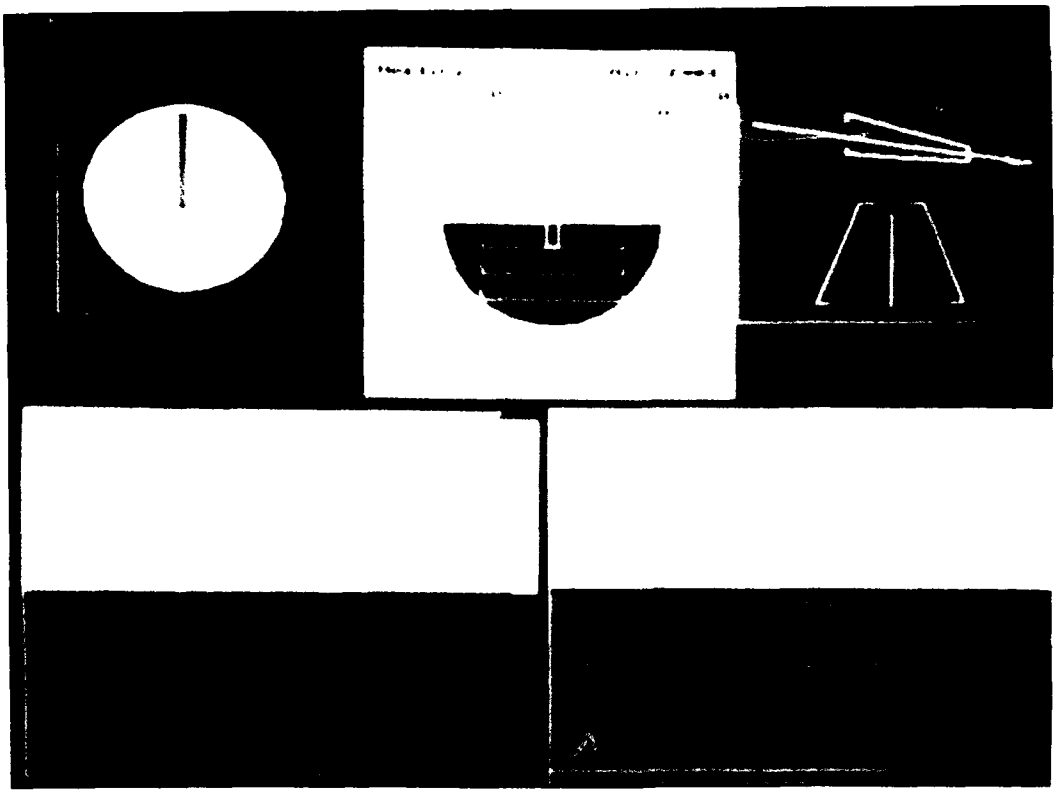


Figure 3.12 Example of Display Test Manager Utilization: Take-off Run, Part 1

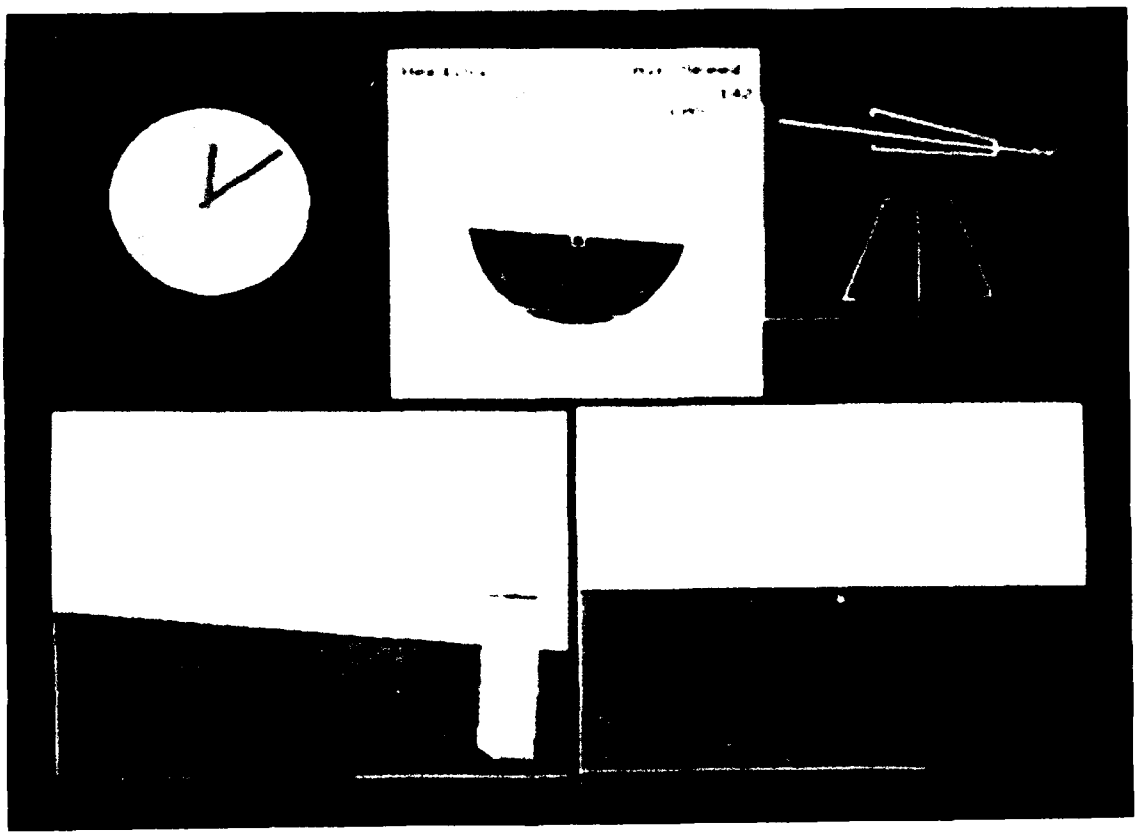


Figure 3.13 Example of Display Test Manager Utilization: Take-off Run, Part 2

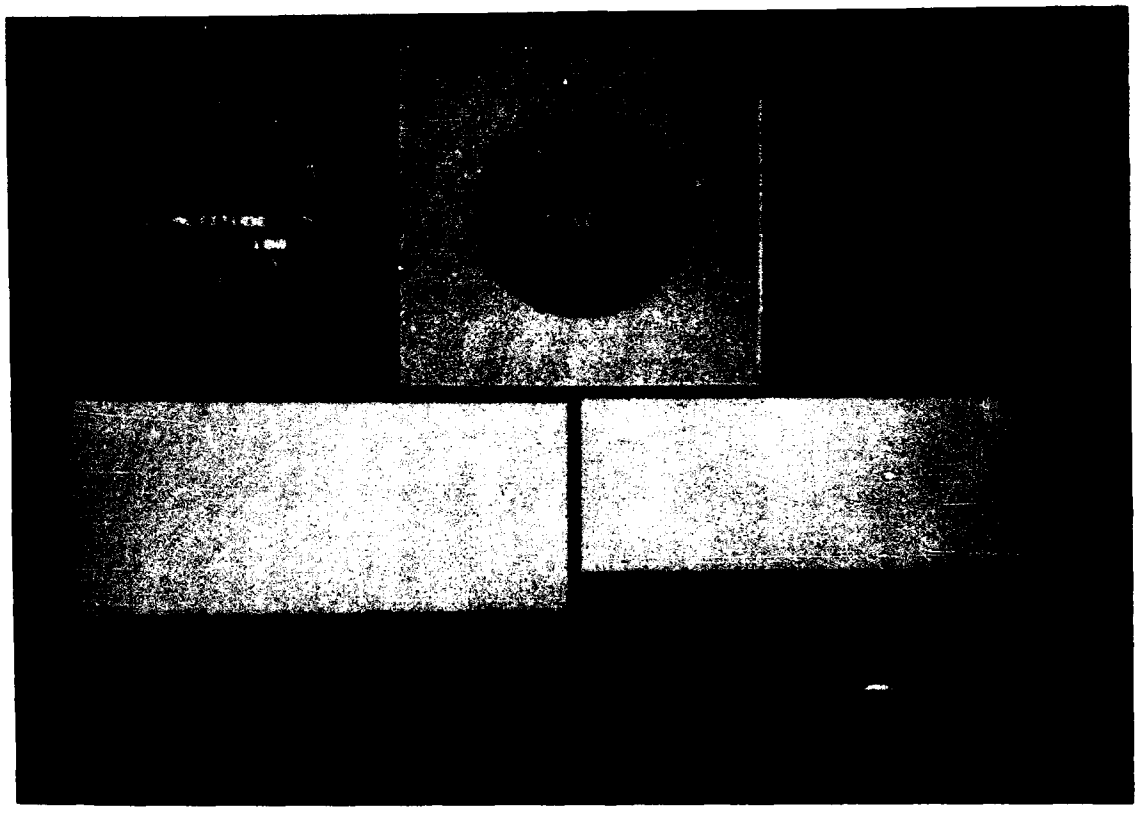


Figure 3.14 Example of Display Test Manager Utilization:
Take-off Run, Part 3

of the PHIGS standard. The other displays in this sequence of figures are also animated by the aircraft simulator. This is an example of the possible implementation of the concept of a windowed display system in future cockpit displays.

Fourth, a Display Program Generator application allows the user to produce a self-contained set of files which will implement the display program running outside AGSSS. Two modes of generation are provided: the generation of a "stand-alone" display program, and the generation of a display program which will interface to an outside data interface such as an aircraft data bus or a simulator. This module will access the generated SOFA code through the Actions Editor and the generated graphics structures through the Graphics Editor. The Display Program Generator outputs two types of files: 1) Ada source code files for the display program and 2) Command files in the host computer's command language to allow it to build an executable image from the Ada files.

This portion of the display development process is illustrated in Figures 3.15 through 3.19. Figure 3.15 shows the Display Program Generator Menu invoked from the Display Editor Menu. Upon selection of the choice "generate the display program," the module grabs the graphics description files and the actions description files and generate automatically the files necessary to for the run-time display system outside the AGSSS. Next, Figure 3.16 shows the use of the Workspace Manager menu, one of several auxiliary menus in the system, to invoke the File/Directory Manager. This utility is used to verify the generation of the appropriate executable and command files. This process is illustrated in Figures 3.17 and 3.18. Finally, the software system created by AGSSS is run in the target display system outside the AGSSS environment. This is illustrated in Figure 3.19 which shows the ILS format and an altimeter format connected to an aircraft simulation.

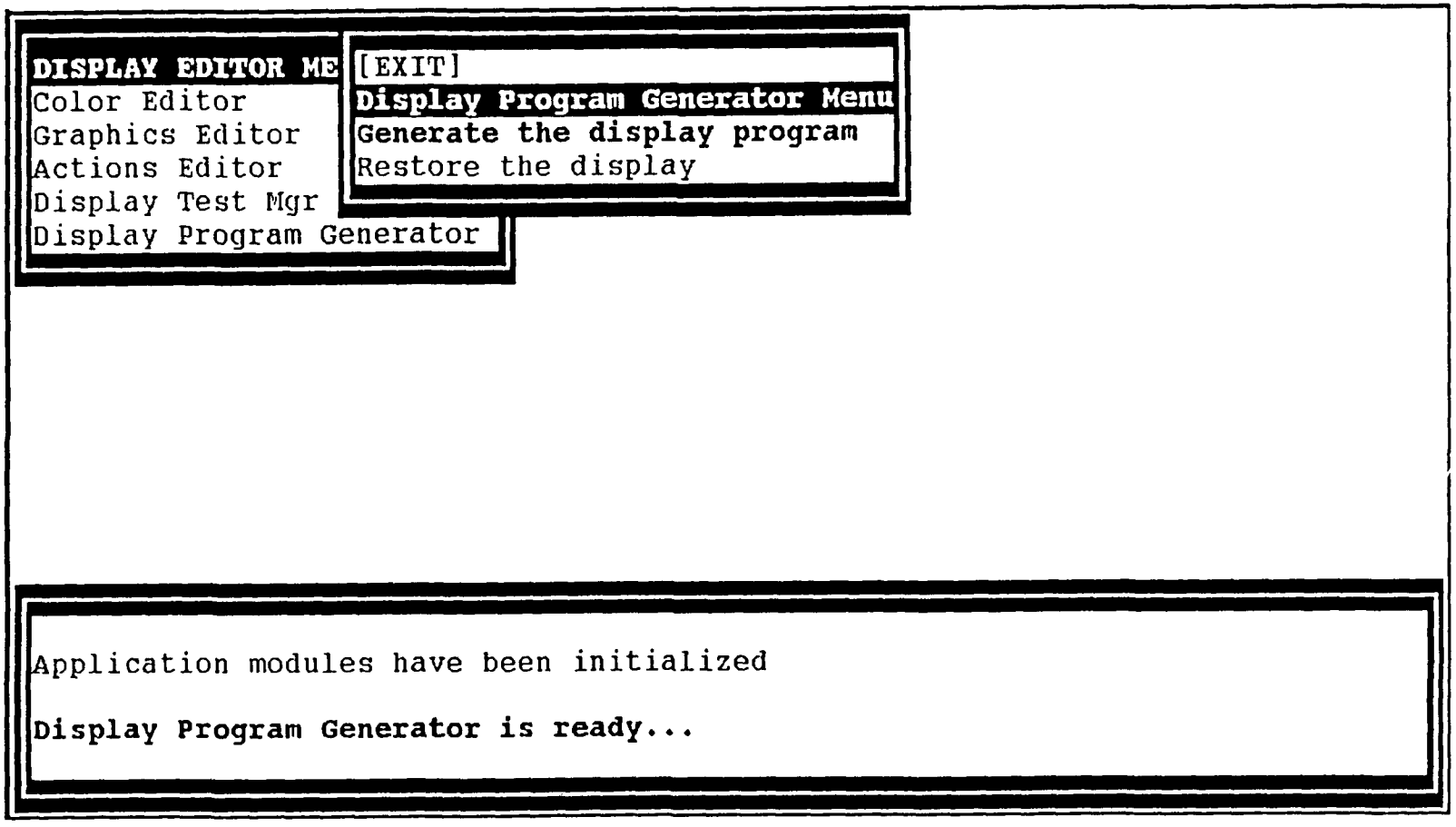


Figure 3.15 Use of the Display Editor Menu to Invoke the Display Program Generator Menu

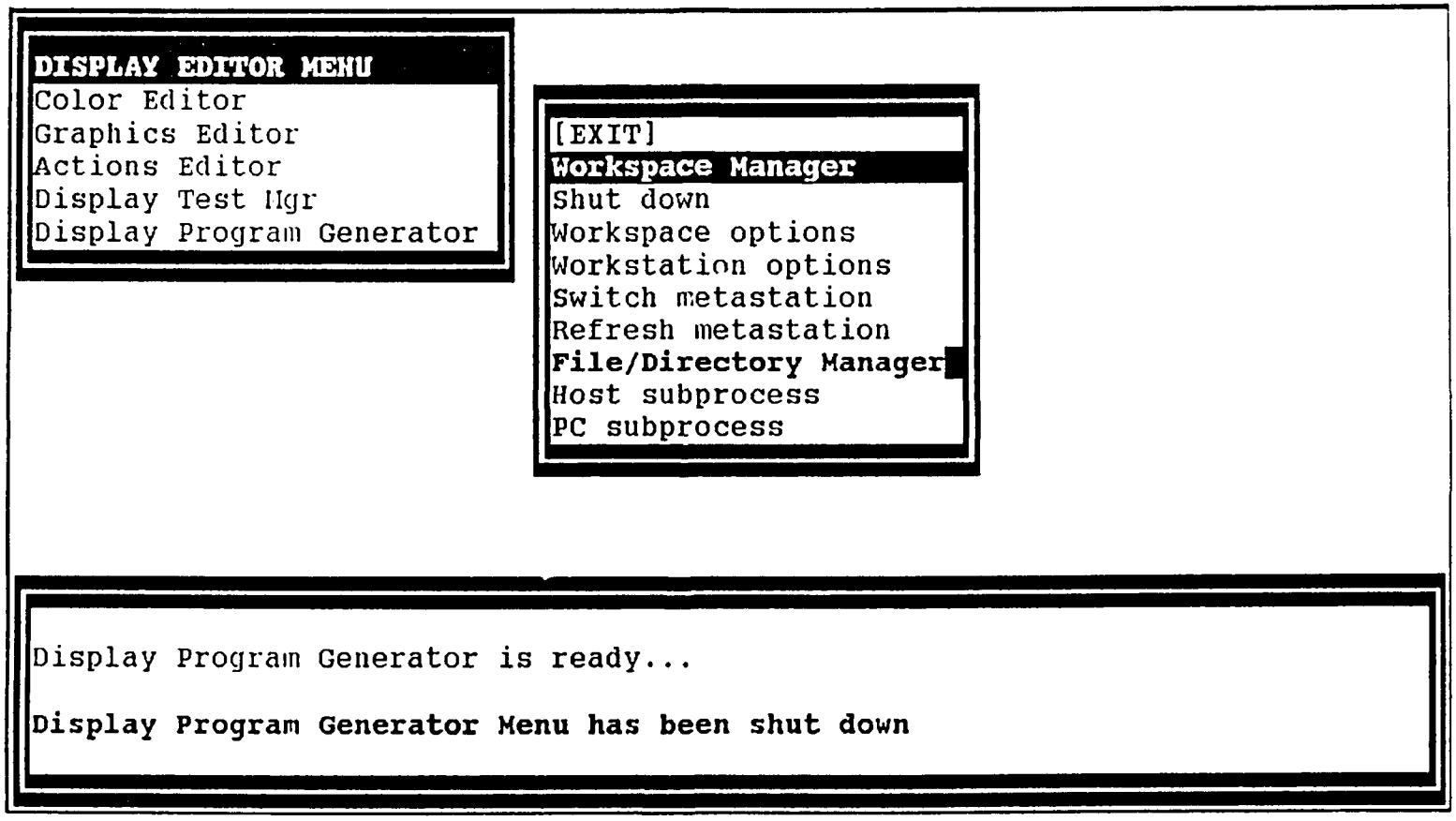


Figure 3.16 Use of the Workspace Manager to Invoke the File/Directory Manager

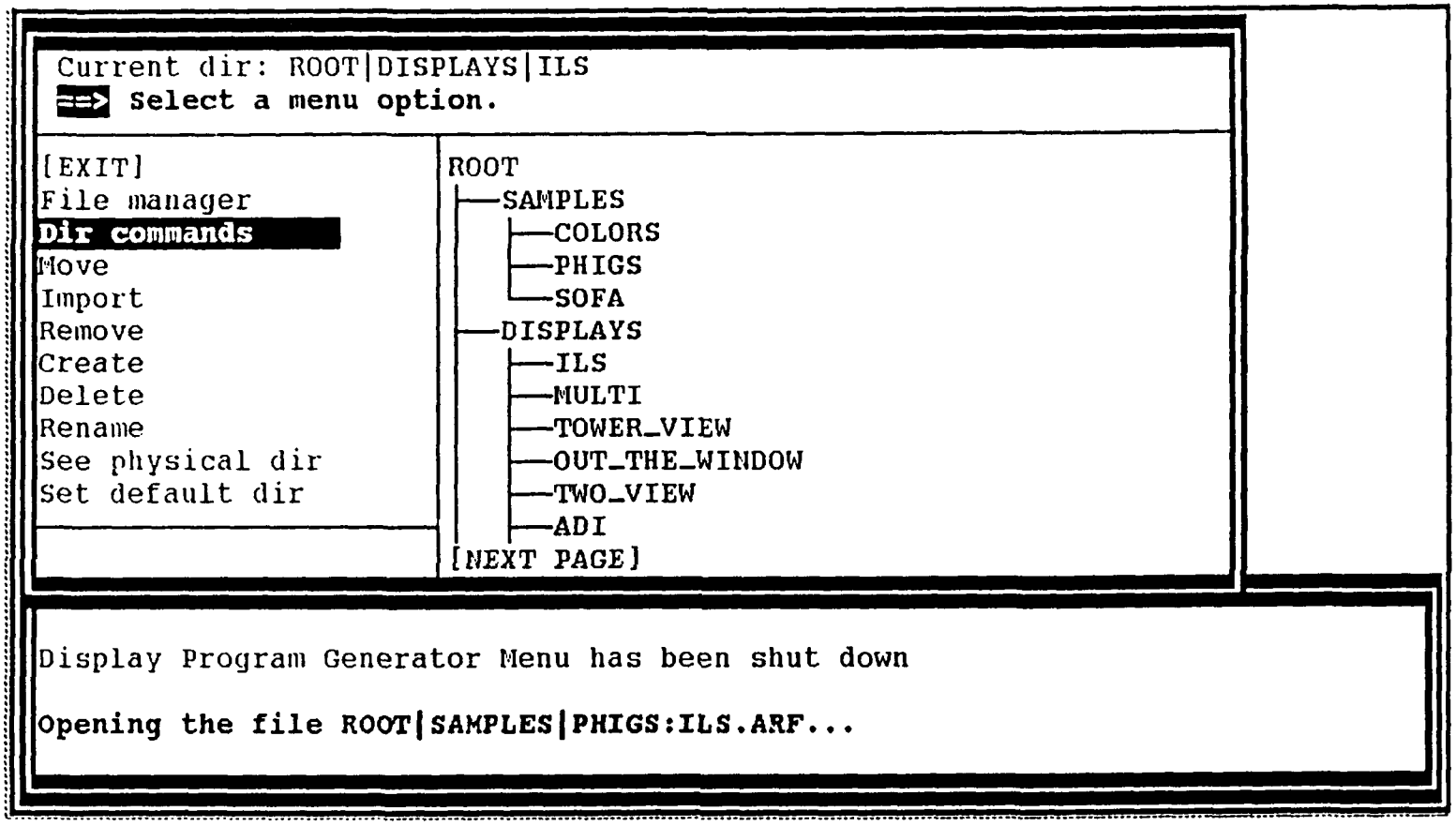


Figure 3.17 Use of the File/Directory Manager to Verify the Generation of the Appropriate Executable Files

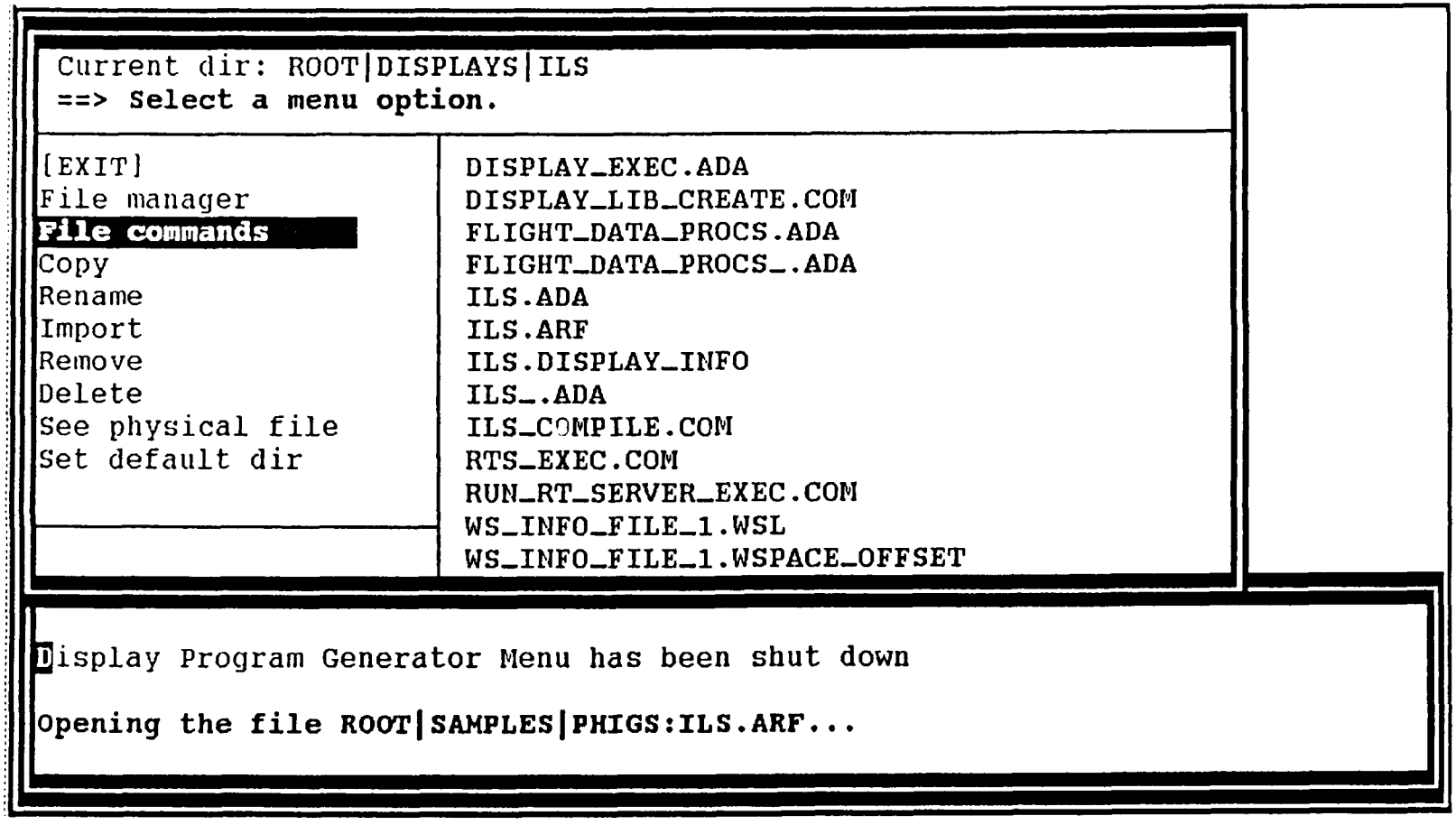


Figure 3.18 Use of the File/Directory Manager to Verify the Generation of the Appropriate Run-Time Command Files

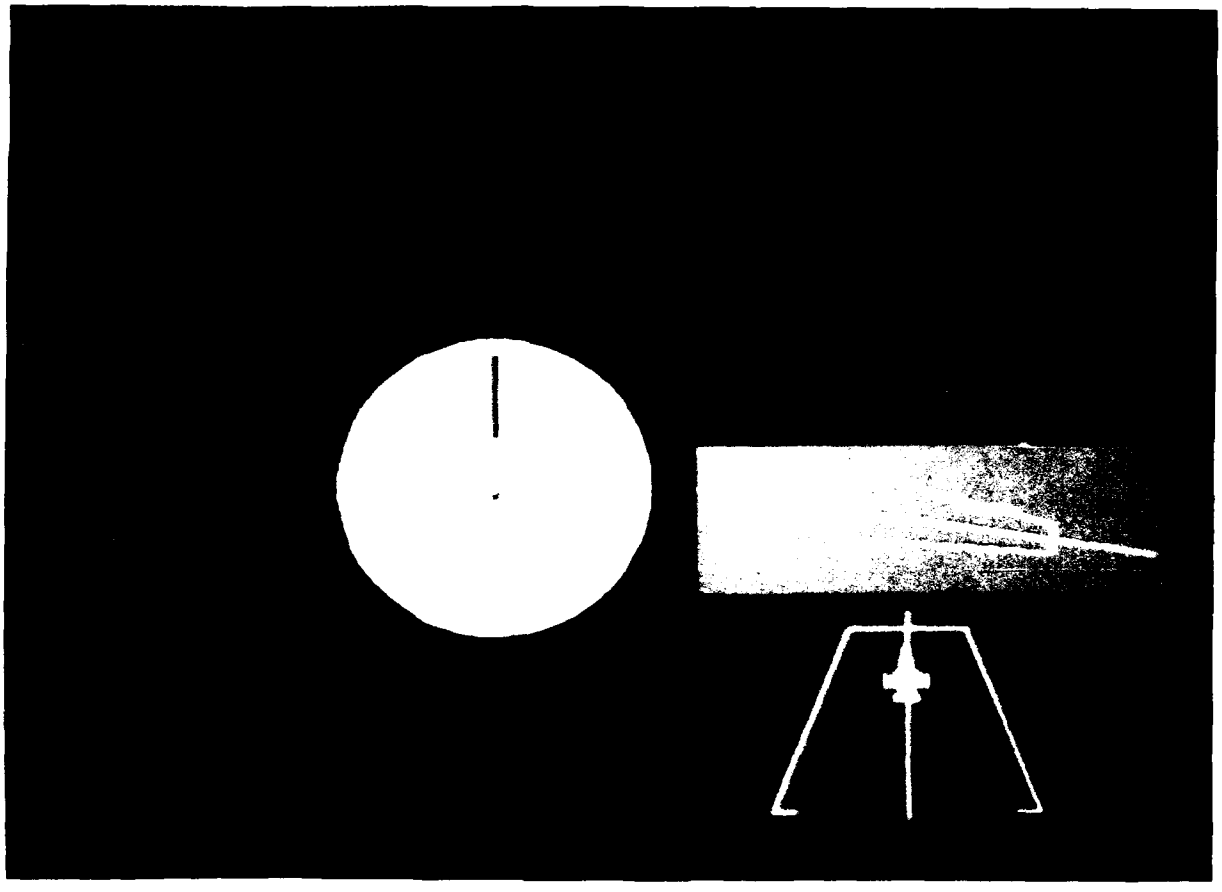


Figure 3.19 Example of AGSSS-Developed Display
Running Outside the AGSSS Environment

4.0 ANCILLARY RESEARCH RESULTS

Some aspects of the AGSSS design used techniques which had not yet been proven. As such, the following are research results derived from the AGSSS effort:

- Use of DIANA as the basis for an integrated Ada development system
- Various extensions to the PHIGS standard
- Development of a portable file manager.
- Use of Ada tasking features to implement user inputs as concurrent finite automata.

These results are described in greater detail below.

4.1 Use of DIANA for an Integrated Ada Development System

DIANA, the Descriptive Intermediate Attributed Notation for Ada, is an abstract representation for Ada programs. It is in use in a number of commercial Ada compilers, notably Tartan and Intermetrics. Specifically, DIANA is an abstract data type capable of describing any correct Ada program.

Because DIANA represents an Ada Program as a data structure, the DIANA representation reflects the structure of the source program. This fact makes DIANA potentially useful for a wide variety of programming tools. In AGSSS, DIANA filled the need for a common internal representation for use in an integrated Ada development environment.

AGSSS had a requirement that it would be able to accept Ada source code from sources other than AGSSS itself. The most sensible approach was to use Ada itself as the specification language for the display formats. The task of AGSSS was thus to provide an integrated development environment for display programs in Ada. The Actions Editor of AGSSS provides

- An incremental parser, which translates Ada source programs into DIANA form
- A source code reconstructor, which translates DIANA back into Ada source
- Syntax-directed editor, which guides the user in the construction and modification of Ada programs
- An interpreter, which runs the Ada/DIANA programs directly, without the conventional compile and link stage
- An interface library, which allows the interpreter to access object code libraries (for example, the interface library that provides an emulation of the run-time environment for the embedded display system)
- A display program generator, which, among other things, determines the compilation order of Ada modules for the target display system.

All these modules make use of the DIANA representation of the Ada display program under development. The syntax-directed editor, while providing a source representation to the user, performs most of its operations on the internal DIANA representation. The interpreter executes by traversing the DIANA data structures and performing the indicated actions. The interface library converts subprogram references in DIANA to calls to object code routines. The display program generator makes use of the DIANA list of compilation units to determine how to build the target display program.

AGSSS uses DIANA revision 4 from Intermetrics. The subset of Ada supported by AGSSS is quite large; however, it does not support tasks, generics, or discriminant records.

4.2 Extensions to the PHIGS Graphics Standard

The Programmer's Hierarchical Interactive Graphics System, and ANSI and ISO standard, provides a means of define graphics data which are easily modified and updated. For this reason, PHIGS is suitable both for interactive display definition, as well as for real-time display

generation. To support both of these activities, we wrote an implementation of PHIGS in Ada. We found it necessary to add features to PHIGS which were not addressed in the standard. Of course, making changes to a standard reduces portability, one of the main reasons for the use of standards in the first place. In some areas, PHIGS provides approved methods for adding extensions. Our approach, designed to maximize portability, was as follows:

1. Where possible, confine the PHIGS extensions to a package surrounding or external to PHIGS rather than to PHIGS itself.

2. Where it is necessary to extend PHIGS itself, use the approved extension mechanisms if possible.

3. Since display programs will be ported more often than AGSSS itself, itself, confine the generated display programs to the PHIGS standard without extensions, as far as possible. Extensions contained in a package surrounding or external to PHIGS are permissible, since these would not involve any changes to PHIGS itself.

Extensions to PHIGS carried out according to these rules include:

- Concurrent PHIGS for parallel access by multiple Ada tasks
- A PHIGS interface package, for use in real-time display generation
- Names for structures, elements
- Private Structure stores
- Replication element type
- Additional input classes.

In observation of rule 1) above, we created packages to surround PHIGS, where we implemented most of our extensions. Because of the differing requirements for PHIGS within the AGSSS tool, and PHIGS as a real-time display system, there are two such packages.

The package for AGSSS PHIGS is called CONCURRENT_PHIGS. Since AGSSS has multiple Ada tasks which could potentially access PHIGS, we needed to insure that each could use PHIGS without interfering with each other.

This was accomplished by creating a task with entries for each routine in the PHIGS library. The entries all have an additional argument containing information about the context of the calling task. If all tasks access through the CONCURRENT_PHIGS task, then conflicts are avoided.

In the generated real-time display, concurrency is not a requirement. In this case, we provide a package called PHIGS_IFACE (PHIGS Interface), which simply maps standard PHIGS calls to the particular PHIGS implementation upon which the display is running. This scheme takes care of the many discrepancies which can be found between different vendors' versions of PHIGS.

Following rule 2) above, CONCURRENT_PHIGS contains several enhancements to PHIGS which are implemented outside of PHIGS itself. The most important of these are private structure stores and names for structures and elements.

In PHIGS, display list commands are organized into groups called PHIGS structures. Individual commands are called structure elements. All structures reside conceptually in a common area called the Centralized Structure Store (CSS). This arrangement corresponds roughly to a large FORTRAN program in which all variables reside within a single common block. In the concurrent case, there is extreme danger that competing tasks will corrupt each other's data. For this reason, CONCURRENT_PHIGS offers each task only a select view of the CSS. That is, the user can create any number of virtual CSS's, whose contents are invisible to the others. Such a virtual CSS is called a private store. This is accomplished by mapping the structure identifiers in the CSS to unique identifiers in the private store. All entries in the concurrent PHIGS task require a store identifier in addition to the standard PHIGS parameters.

Each private store contains a mapping of structure names (Ada STRING types) to structure identifiers (integers). This allows the interactive workstation to display structures by name rather than by number.

Additionally, AGSSS provides names for structure elements. However, this is accomplished using a feature of PHIGS itself, namely the PHIGS Application Data Element. This element contains a character string which we use to encode the name of the element which follows in the structure's element list.

PHIGS provides two extensions by which the implementor can add his or her own structure element types, called Generalized Structure Element and Generalized Drawing Primitive. Any such extensions, while permitted by PHIGS, are not, in general, portable to other PHIGS implementations. Our PHIGS implementation offers one generalized drawing primitive, called REPLICATE_STRUCTURE. This element has the effect of an EXECUTE_STRUCTURE element followed by a local modelling transformation, repeated a specified number of times. It is used to repeat a picture at regular intervals, such as lines of a pitch grid, or tick marks on a linear or circular scale. The possible rotation, translation, or scaling of each repeated image is controlled by the specified modelling transformation. The transformations are cumulative with each iteration.

If the display is targeted to a system which does not support such a generalized structure element, each REPLICATE_STRUCTURE element in the display specification can be replaced by a series of EXECUTE_STRUCTURE and local modelling transformation elements. This can be accomplished automatically within the DISPLAY PROGRAM GENERATOR of AGSSS.

Another extension permitted by PHIGS is in the area of prompt/echo types. Prompt/echoes are the feedback to the user of the current input state. Examples include graphical cursors in response to locator inputs and screen characters in response to text inputs. In addition to the prompt/echo types defined by PHIGS, implementors are free to define their own types. Of course, such types will not port across differing PHIGS implementations. AGSSS PHIGS makes use of some new locator prompt/echo types, which, according to rule 3) above, are not used in the generated real-time displays.

The new locator prompt echo types are:

- PRIMITIVE_LOCATOR - drags all instances of a particular structure element. A specified vertex of a specified instance takes the current locator position.
- PATH_LOCATOR - drags a single instance of a particular structure element. A specified vertex takes the current locator position.
- VERTEX_LOCATOR - drags a vertex of all instances of a particular structure element.
- PATH_AND_VERTEX_LOCATOR - drags a vertex of a single instance of a particular structure element.

In the current implementation, these echoes all use an exclusive-or operator on the pixels of the underlying image. When the echo moves, the old echo is erased by repeating the exclusive-or operation, then the new echo is drawn.

Drawing the echo in the proper position requires structure traversal, applying all the modeling and viewing operations, as well as incorporating the relative translation defined by the locator position. For this reason, these echoes will only be possible under AGSSS PHIGS.

Although these echo types are not portable, it may be possible to achieve similar effects on other PHIGS implementations through other means. On a platform with sufficient graphics throughput, for example, the locator position could be used to edit the structure contents to update the whole picture continuously. The user would then see the end result, rather than a "rubberband" image of the changes.

4.3 Development of a Portable File Manager

Since AGSSS is intended to be highly portable, it is desirable to shield the user from the conventions of the host operating system, as well as from worries about storage devices. AGSSS provides an abstract directory tree and a set of operations on files and directories. Special map files control the mapping of subdirectories and files to

subdirectories and files on the host system(s). Users can manipulate files and directories through a graphical interface which displays the abstract directory tree.

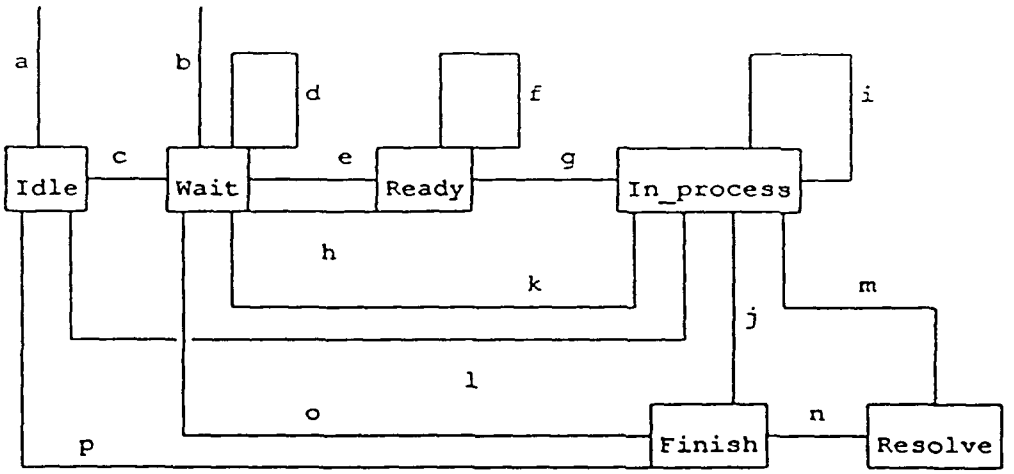
Each subdirectory in the abstract directory tree corresponds to a disk directory somewhere in the system. However, the disk directories need not follow the same hierarchy as the abstract directory tree. The directories can reside on any disk in the system. They may even reside on the PC front end. The file manager operations can transfer files to and from the PC as needed.

4.4 Use of Ada Tasking Features to Implement user Inputs as Concurrent Finite Automata

The idea of modeling user inputs as concurrent finite automata dates back to Jacob, (R 8). Thinking of an input as a finite automaton means that there is a state transition diagram which describes all possible states for the input. Each state can respond to a number of possible events. Each event is met by some action, including a possible change of state. Figure 4.1 shows an example of a PHIGS pick input modelled in this way. Device inputs are concurrent in two senses. The user may be using multiple devices simultaneously, e.g., mouse and keyboard. Second, there may be multiple windows on the screen(s), each with its own inputs. Their state is maintained even as control passes from one window to the next.

In AGSSS, the first type of concurrency is handled by Ada tasking. Each physical input is monitored by a separate Ada task, and all inputs are event driven.

The second type of concurrency is handled by assigning a state transition record to each open input. When an event occurs which might affect a particular input, a procedure is called which looks up the current state from the state transition record, then takes the appropriate action.



The arcs of the graph are as follows:

a: Device is initialized in request mode.
b: Device is initialized in sample or event mode.
c: A begin request call is made for this device.
d: Non-"Ready" input is received.
e: Ready input is received.
f: Ready input continues to be received.
g: "Process" input is received inside the echo area.
h: Data is received which cannot be classified in f or g.
i: Process input continues to be received (inside or outside the echo area). Echo this data.
j: Finish data is received.
k: Abort data is received in sample or event mode.
l: Abort data is received in request mode.
m: Finish data is received and is sent down to graphics mgr for resolution (PHIGS locator)
n: The resolved information is received from graphics mgr
o: Data is sent to logical device in sample or event mode.
p: Data is sent to logical device in request mode.

Figure 4.1 Drag Finite State Machine

5.0 CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

At the beginning of the project, there were concerns about the real-time performance of the display programs to be generated by AGSSS. These concerns arose mainly from two issues: 1) the comprehensive and complex nature of the Ada programming language and 2) because PHIGS graphics rendering is achieved by interpreting its graphics commands. In the case of Ada, the concern was not whether Ada and Ada/PHIGS could be used in real-time applications, but whether or not the overhead penalty associated with its execution would be excessive or the efficiency of the generated code would be acceptable. In the case of PHIGS, the concern was that, as a rule, structure traversal (required by interpretation) takes more time than direct execution of equivalent code. Concerning Ada, the AGSSS automatic Code Generator was designed to take advantage of Ada's many general-purpose features and to support the tailoring of the generator code for optimal real-time execution in a target display system. Concerning PHIGS rendering, the AGSSS design included provisions to implement the PHIGS rendering pipeline in the machine code of the target display generator and thus enhance the real-time performance of the graphics rendering process. Also, the emergence of graphics display generators optimized for execution of PHIGS procedures may obviate the need for this implementation in the future.

In general, AGSSS has met or exceeded its design goals. Productivity associated with the process of developing cockpit displays and its enabling software has improved tremendously both in quantity and quality. The specialized knowledge required heretofore to work in this area has been reduced. Also, the system generates display programs which should be portable to other Ada/PHIGS environments.

Preliminary results indicate that productivity has increased by a factor of at least 10 over that obtained with the conventional method (FORTRAN/RAP) of developing display software for the ITB Facility Display Generation System (DGS). Furthermore, it is estimated that the integrated environment provided by AGSSS will improve productivity by a factor of 10 over hand-coding the same application in Ada/PHIGS and that further increases are possible. The Graphics Editor implementation is very thorough and has the potential for many more functions. It has certainly proved its worth as a rapid prototyping tool supporting the iterative development process. The development of Ada programs has also benefited from the AGSSS implementation. For example, using the Actions Editor Ada program turn-around has improved between 10 and 50 times over the standard approach.

The potential is there for substantial quality improvements obtained by exploiting the iterative development process, the use of PHIGS, and the porting of AGSSS and its products to newer graphics platforms. Concerning the amount of specialized knowledge required to produce a cockpit display, significant reductions have been obtained by supporting the graphics specifications at a much higher level and implementing AGSSS to act as a guide with respect to the specifications of the actions in Ada. Finally, indications are that, with the exception of the generalized structure elements and the color specifications, the code generated by AGSSS will be 95% to 100% portable to other compatible platforms.

Furthermore, the targetability of the products of AGSSS has been demonstrated, albeit on a preliminary basis, by the successful target of code produced by AGSSS to other display system environments. Figures 5.1 and 5.2 illustrate this accomplishment. Figure 5.1 illustrates the display software generated as described in Section 3.3 running in a VAXStation 3100. Figure 5.2 illustrates the same software running in a SUN/4 SparcStation 370. In both cases, we have interfaced to the display generation process through the local (DEC or SUN) implementation of PHIGS. These preliminary results are very encouraging.

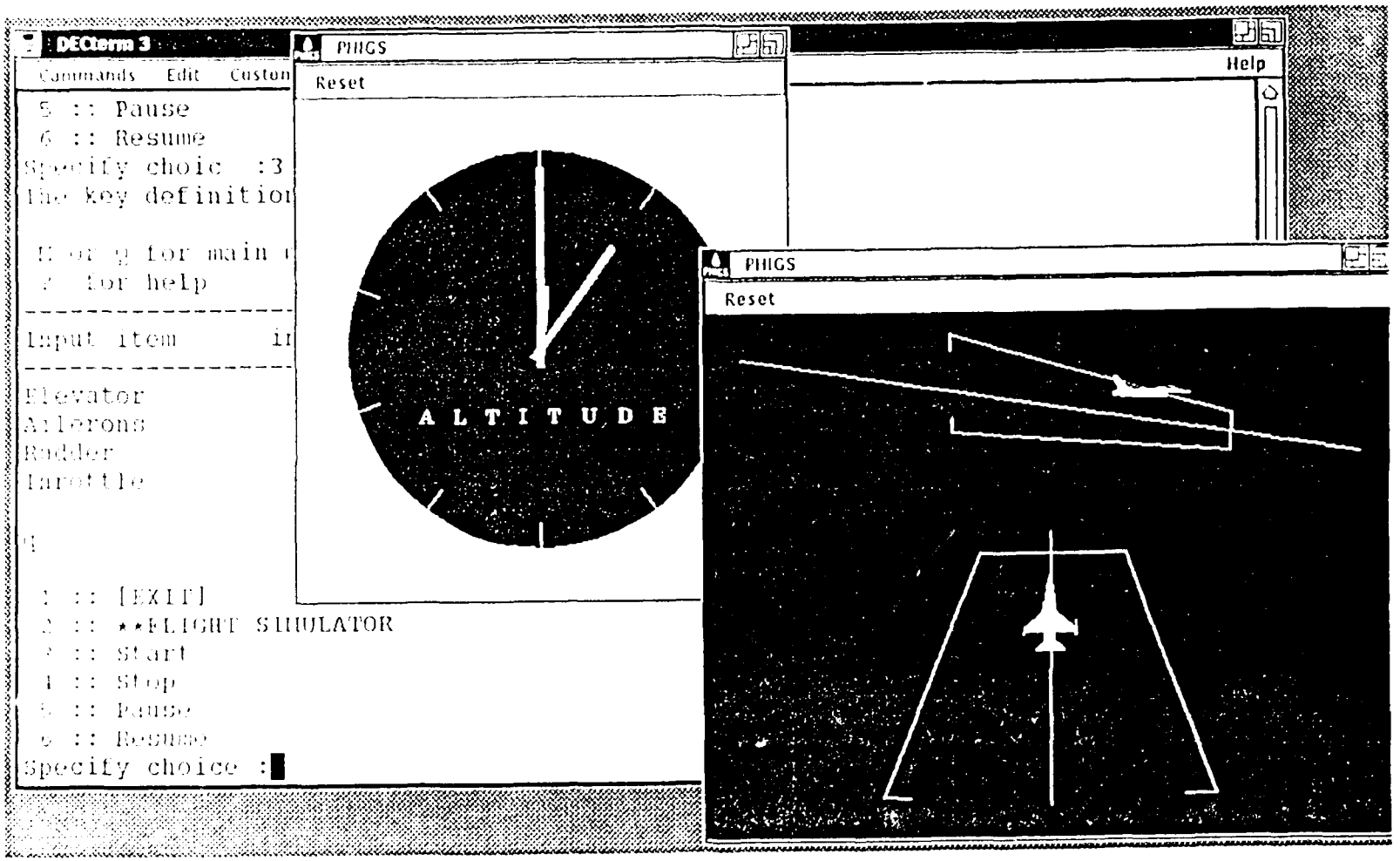


Figure 5.1 Example of AGSSS-Generated Display Software System Targeted to a VAXStation 3100

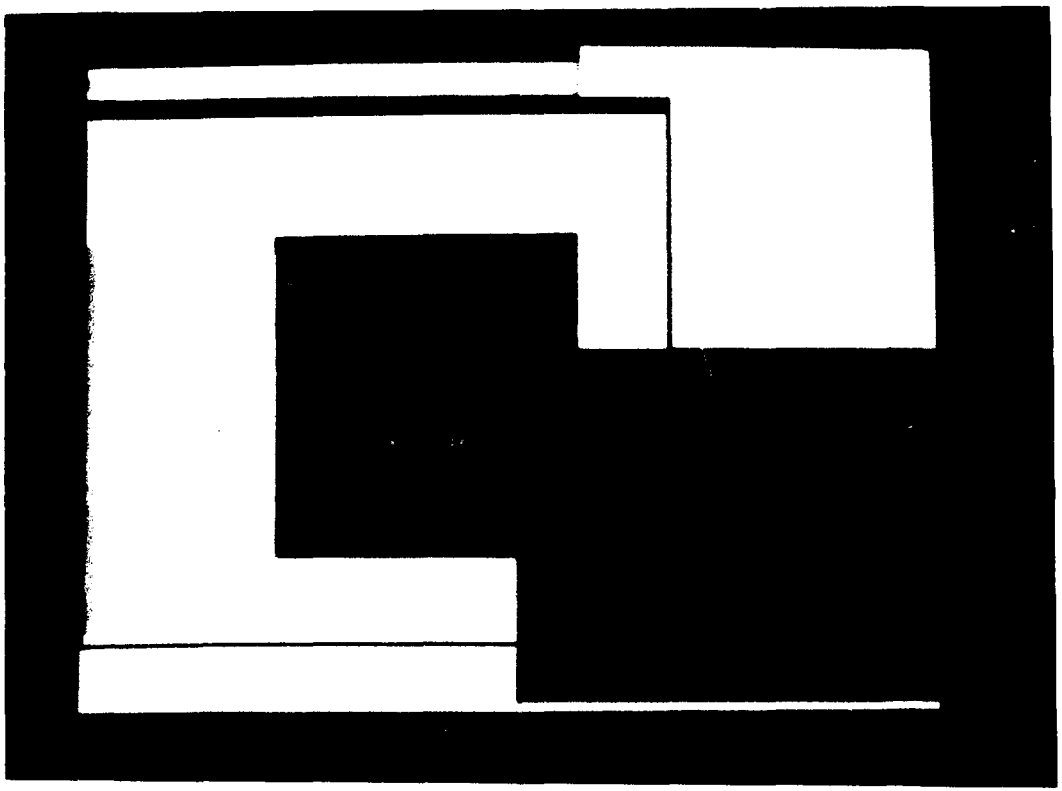


Figure 5.2 Example of AGSSS-Generated Display
Software System Targeted to a SUN/4 SPARCStation 370

5.2 Recommendations

The AGSSS was earmarked for implementation and demonstration in the display generation system (DGS) of the ITB Facility. This display system, based on the Adage 3000 PDG, has been in operation since 1985 with current plans calling for its replacement, in the not-too-distant future, with a DGS based on one of today's high-performance workstations. This workstation will most likely be one that supports PHIGS and PHIGS PLUS.

Based on the results obtained with AGSSS, near-term recommendations include the porting of the tool to one or more of today's high-performance graphics workstations, especially the one chosen to upgrade the ITB Facility's DGS with. In addition the products of AGSSS should be targeted to the workstation of choice and the targeting strategy developed to exercise the application code produced by AGSSS in as many DGS as possible. Also subset configurations of AGSSS should be considered for other avionics applications. Specific enhancements to components of AGSSS should also be considered. These could include the addition of high-level primitives and PHIGS PLUS (PHIGS Plus Lumiere Und Surfaces) enhancements to the Graphics Editor; support for PDL and document generation in the Actions Editor; and provision of non-aerodynamic motion control in the Display Test Manager.

Longer term recommendations are based on exploiting the fact that the AGSSS has been designed and implemented with a great deal of modularity. For example, the Kernel, the Device Interface, and the Ada Tools components of the AGSSS can be viewed as forming the basis for a generic Ada/PHIGS workstation which may be used to develop software for other, nongraphical, embedded applications. Furthermore, an implementation strategy should be followed that merges both the AGSSS windowing functionalities into those of X Windows, and the AGSSS PHIGS graphics with those of the emerging PEX (PHIGS Extensions to X) standard. This approach would promote the widest utilization of this productivity-enhancement tool.

Cockpit displays will continue to include more and more information with ever-increasing sophistication as the technology and the display designer's imagination continue to leap-frog each other. Complex displays such as the one depicted in Figure 5.3 (Pathway-in-the-Sky (PITS)) will become common place in the not-too-distant future. We believe that the Airborne Graphics Software Support System (AGSSS) will provide an important, robust, absolence-proof, tool for display designers to work with for many years to come.

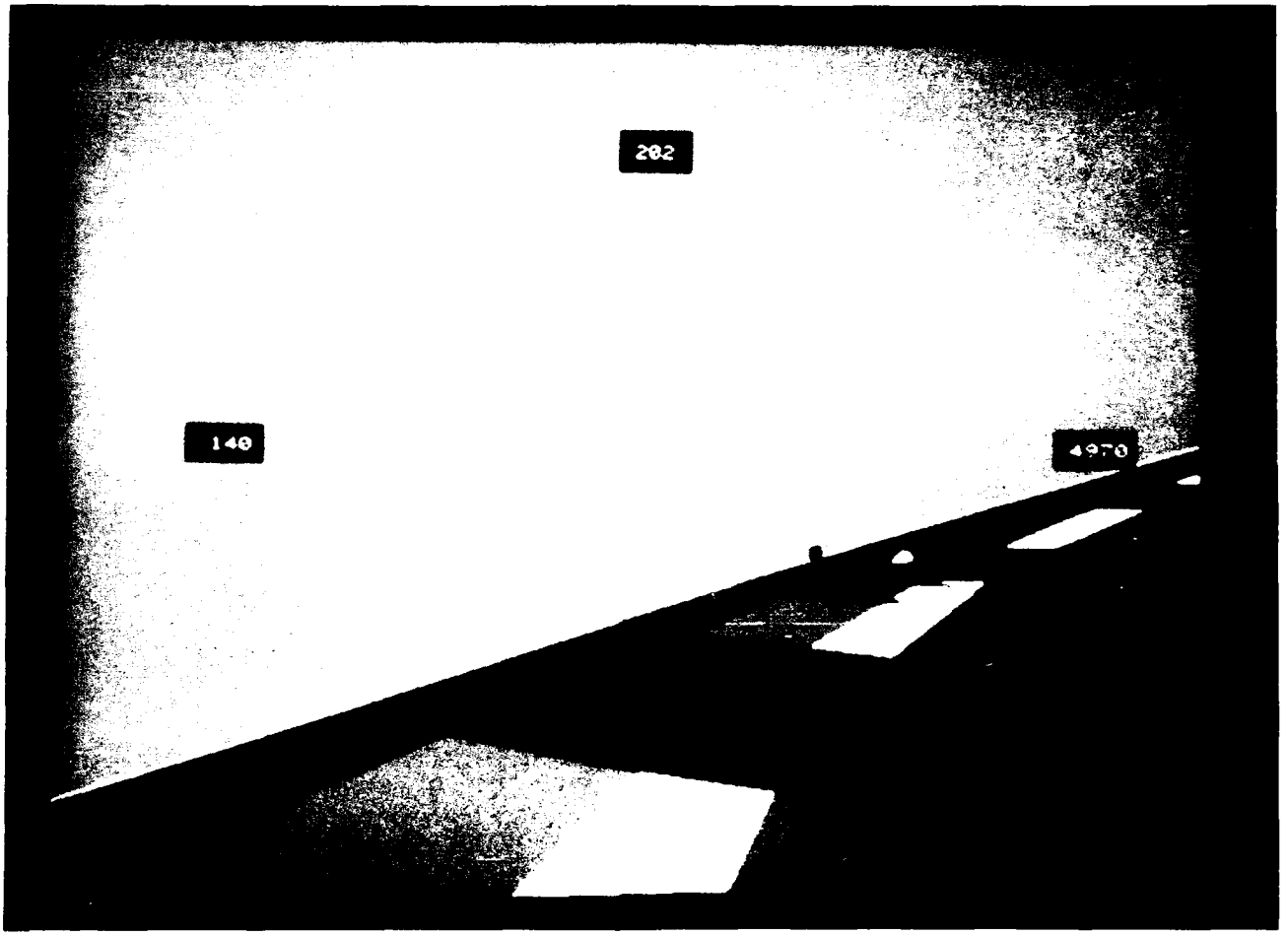


Figure 5.3 Example of Complex Display Format
Expected to be Used in the Cockpit of Future Aircraft

6.0 REFERENCES

1. Montoya, et al, "An Interactive Graphics Editor for Computer-Generated Cockpit Displays." Proceedings of the IEEE/AIAA/NASA 9th Digital Avionics Systems Conference, October 15-18, 1990, Virginia Beach, Virginia.

2. American National Standards Institute (ANSI). "Computer Graphics--Programmer's Hierarchical Interactive Graphics System (PHIGS) Functional Description," September 26, 1988. X3.144-1988.

3. ANSI/MIL-STD-1815A-1983, "Reference Manual for the Ada Programming Language," February 17, 1983.

4. ANSI document X3H3/86-43R1, Working papers for dpANS X3.144.3-198x, "Computer Graphics--Programmers Hierarchical Interactive Graphics System (PHIGS) Binding to Ada," May 1987.

5. Montoya, et al, "An Ada-based, Portable Design Workstation for Computer-Generated Cockpit Displays." Proceedings of the IEEE/AIAA/NASA 9th Digital Avionics Systems Conference, October 15-18, 1990, Virginia Beach, Virginia.

6. Anon., RDS 3000 User's Guide. ADAGE, Inc., Document no 10-301-095-10A, Billerica, Massachusets.

7. "Software Detailed Design Document for the Airobrne Graphics Software Support System (AGSSS)." Contract No. F33615-87-C-1531 CDRL No. 9. Prepared by Research Triangle Institute, RTP, NC, January, 1991.

8. Jacob, Robert J.K., "A specification language for direct-manipulation user interfaces." ACM Transactions on Graphics, v. 5 no. 4, October, 1986.

Bibliography

9. Aho, A. V. and J. D. Ullman, "Principles of Compiler Design," Addison-Wesley Publishing Co., London, 1979.

10. Booch, Grady, "Software Engineering with Ada," The Benjamin Cummings Publishing Company, Inc., Menlo Park, California, 1987.

11. Brown, G. P., et al., "Program Visualization: Graphical Support for Software Development," IEEE Computer 18, no. 8, August 1985

12. Clark, J., Measuring Basic 3-D Graphics Performance, Computer Graphics Today, pp. 26-27, Volume 5, Number 2, February 1988.

13. Foley, J. D. and A. Van Dam, "Fundamentals of Interactive Computer Graphics," Addison-Wesley Publishing Co., London, 1983.

14. Fuchs, H., J. Poulton, et al., "PIXEL-PLANES, A Parrallel Architecture for Raster Graphics." Pixel-planes Project Summary, Department of Computer Science, University of North Carolina at Chapel Hill, August 1986.

15. Hasker, R. W., J. S. Edmunson, and M. R. Fritsch, "The Automated Programming of Electronic Displays." AFWAL-TR-86-3046.

16. LoPiccolo, P. J., Tools Chart High-End Workstations. Engineering Tools, pp. 80-95, volume 1, number 1, February 1988.

17. Montoya, R. J., J. N. England, J. J. Hatfield, and S. A. Rajala, "An Advanced Programmable and Reconfigurable Color Graphics Display System for Crew Station Technology Research." AIAA/IEEE Fourth Digital Avionics System Conference, St. Louis, Missouri, November 17-19, 1981.

18. Torborg, J. G., A Parallel Processor Architecture for Graphics Arithmetic Operations. Computer Graphics, pp. 197-204, Volume 21, number 4, July 1987.

19. IST91398401 Mission Software Controls and Displays Interface Control Document.

GLOSSARY

Ada	Programming Language
AGSSS	Airborne Graphics Software Support System
ALC	Automated Layout Center
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
BIOS	Basic Input Output System
BPS	Bipolar Processor Set
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CFG	Configuration
CGA	Color Graphics Adapter
CISC	Complex Instruction Set Computer
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSE	Center for Systems Engineering
CIG	Computer Image Generation
CRT	Cathode Ray Tube
DC	Direct Current
DEC	Digital Equipment Corporation
DEV	Device
DCL	Digital Command Language
DIANA	Descriptive Intermediate Attributed Notation for Ada
DID	Data Item Description
DIR	Directory
DMA	Direct Memory Access
DoD	Department of Defense
DOS	Disk Operating System
DRD	Data Requirements Document
EGA	Enhanced Graphics Adapter
ESC	Escape
FORTRAN	Formula Translator
GKS	Graphical Kernel System
HIRES	High resolution
HLL	High Level Language
HSD	Horizontal Situation Display
HWCI	Hardware Configuration Item
Hz	Hertz
HWCI	Hardware Configuration Item
IDL	IKONAS Display Language
IGE	Interactive GRAPHICS EDITOR
IKASM	IKONAS Assembler
INFO	information
INIT	Initialize
ITBF	Integrated Test Bed Facility
I/O	Input/Output
KBD	Keyboard
KW	Kilowords
K	Kilo-, thousand
LaRC	Langley Research Center
LLCSC	Low Level Computer Software Component

LORES	Low Resolution
M	Mega-, million
MAX	Maximum
NASA	National Aeronautics and Space Administration
NDC	Normalized Device Coordinates
NOEL	Node Oriented Editor Language
NTSC	National Television Standards Commission
PATTI	Programmers Attributed Text Interface
OS	Operating System
PC	Personal Computer
PDG	Programmable Display Generator
PHIGS	Programmer's Hierarchical Interactive Graphics System
PIXEL	Picture Element
PRDA	Program Research and Development Announcement
RAP	Real-Time Animation Package
REC	Record
RISC	Reduced Instruction Set Computer
RTI	Research Triangle Institute
SALC	Simplified Automated Layout Center
SCCRS	Software Configuration Control and Reporting System
SPEC	Specification
STLDD	Software Top Level Design Document
STD	Standard
SUM	Software User's Manual
TI	Texas Instruments
TLCSC	Top Level Computer Software Component
USAF	United States Air Force
USN	United States Navy
VAR	Variable
VMS	Virtual Memory System
WC	World Coordinates
WL	Wright Research and Development Center
WS	Workstation
WSPACE	Workspace
XMIT	Transmit

U.S Government Printing Office 1991—648-127/62057